BOURNEMOUTH UNIVERSITY

Fairness for Resource Allocation in Cloud Computing

by

Hamed Hamzeh

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Science and Technology

2020-08-05

Declaration of Authorship

I, HAMED HAMZEH, declare that this thesis titled, 'Fairness for Resource Allocation in Cloud Computing' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Hamed Hamzeh

Date: 05/08/2020

Bournemouth University(BU)

Abstract

Faculty of Sciecne and Technlogy
by Hamed Hamzeh

Resource allocation with fairness, considering multiple types of resources has become a substantial concern in state-of-the-art computing systems. Accordingly, the rapid growth of cloud computing has highlighted the importance of resource management as a complicated and NP-hard problem. Unlike traditional frameworks, in modern data centers, incoming jobs pose demand profiles, including diverse sets of resources such as CPU, memory, and bandwidth over multiple servers. Therefore, the fair distribution of resources respecting such heterogeneity has appeared to be a challenging issue. In this thesis, the major research gaps are identified, associated with recent solutions in resource allocation and scheduling with fairness in cloud computing. The recent developments have satisfied some desirable fairness properties such as sharing-incentive, Pareto-efficiency, envy-free, and strategy-proof in general perspective. However, some of these promising features have not been satisfied for some users with demands dominated on a particular resource type that cause an allocation to be intuitively unfair. First, recent approaches have ignored the boundaries of fair-share and resource demands in allocation decisions. Additionally, those approaches have fallen short in prioritizing tasks, considering different resource types in scheduling time that may result in increasing the response time for some users. Second, the recent developments have considered the equalization only for dominant resources that could be an obstacle against utility maximization for a certain number of users. Besides, there is no specific measure for evaluating the fair distribution of resources among tasks with dominant and non-dominant resources, in single and heterogeneous server settings. Third, it is still unclear how the number of dominant resources in multiple servers, considering a specific resource type may affect the Pareto-efficiency and sharing incentive properties. Fourth, investigating the fairness problem, taking into account multiple resource types seems like a missing point in the Kubernetes environment. This significant issue may increase the response time due to a considerable number of pod evictions. This thesis seeks to address these substantial gaps. First, a new mechanism is introduced to calculate shares among users, considering a fair-share function that solves a utility maximization problem. Furthermore, a new queuing mechanism is proposed to reduce response time for incoming tasks dominated on different resource types. Second, to address the fair distribution of resources among users, a fully-fair allocation algorithm is presented as well as a new fairness measure for multi-resource environments. Third, to tackle issues concerning Pareto-efficiency and sharing-incentive in heterogeneous servers, a novel task scheduling mechanism is introduced. Fourth, a new model is suggested to tackle the fairness problem in the

Kubernetes framework to reduce the number of pod evictions in scheduling time. The experiments conducted in the Cloudsim framework, using randomly generated workloads and Google workload traces show that our proposed algorithms achieve higher utilization of resources as well as fairness compared to DRF. Moreover, the evaluations indicate that the proposed algorithms satisfy desirable fairness properties.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Dr. Sofia Meacham for the continuous support of my Ph.D research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better Supervisor for my Ph.D.

I would like to also thank my other Supervisors, Prof. Keith Phalp, and Prof. Angelos Stefanidis who helped me a lot with their extensive knowledge in this Journey.

Also, I would like to express my sincere gratitude to research and innovation department in British Telecom especially Dr.Kashaf Khan, Dr. Detlef Nauck, and Dr. Botond Virginas who guided me regarding the technical aspects of my thesis as without their precious support it would not be possible to conduct this research.

Besides my Supervisors, I would like to thank the rest of my thesis committee, Dr.Thanassis Tiropanis and Dr.Francois Siewe for their insightful comments and encouragement, but also for their questions which persuaded me to widen my research from various perspectives.

Last but not the least, I would like to thank my family, my dear Wife, my parents and my brother, for supporting me spiritually throughout writing this thesis and my life in general. ...

Contents

Declaration of Authorship i

Acknowledgements iv

List of Figures x

List of Tables xiii

Abbreviations xiv

Symbols xv

1 Introduction 1
- 1.1 Fairness in computer science . . . 1
- 1.2 Fairness properties . . . 1
  - 1.2.1 Envy-free . . . 2
  - 1.2.2 Pareto-efficiency . . . 2
  - 1.2.3 Strategy Proof . . . 3
  - 1.2.4 Sharing-incentive . . . 3
- 1.3 Efficiency and non-wasteful allocation . . . 4
- 1.4 Fairness in cloud computing . . . 5
- 1.5 Trade-off between the fairness and efficiency in cloud computing . . . 7
- 1.6 Intuitive fairness in cloud computing . . . 8
- 1.7 Research problems . . . 11
  - 1.7.1 Not consider the fair-share boundaries in resource allocation decisions . . . 11
  - 1.7.2 Not employ the queuing isolation in scheduling time . . . 11
  - 1.7.3 Not consider fully fair resource allocation, among tasks with different resource types . . . 11
  - 1.7.4 Not consider the number of dominant resources in multi resource scheduling in cloud computing . . . 12
  - 1.7.5 Not capture the fairness in Kubernetes . . . 12
- 1.8 Research questions . . . 12
- 1.9 Aims and objectives . . . 13
  - 1.9.1 Studying resource allocation with fairness in cloud computing . . . 13
  - 1.9.2 Investigating the existing problems associated with Dominant Resource Fairness(DRF) and its recent developments . . . 14

1.9.3 Exploring the fairness problem in Kubernetes . . . 15
1.9.4 Assigning incoming tasks fairly among servers with heterogeneous settings in cloud computing . . . 15
1.10 Research methodology . . . 15
1.11 Original contributions . . . 16
1.12 Publications . . . 17
1.13 Thesis organization . . . 18

2 Background and literature review **20**
2.1 Introduction . . . 20
2.2 Background . . . 21
2.2.1 Max-Min fairness . . . 21
2.2.2 Weighted Max-Min fairness . . . 23
2.2.3 Proportional fairness . . . 23
2.2.4 Alpha fairness . . . 24
2.2.5 Quantitative fairness measure . . . 24
2.2.6 Cloud Computing . . . 25
2.2.6.1 IaaS . . . 26
2.2.6.2 PaaS . . . 26
2.2.6.3 SaaS . . . 27
2.2.6.4 Private Cloud . . . 27
2.2.6.5 Public Cloud . . . 28
2.2.6.6 Hybrid cloud . . . 28
2.2.7 Fairness in resource allocation and scheduling in cloud computing . 29
2.2.7.1 Multi-resource system setting . . . 30
2.2.7.2 Multi Resource allocation setting . . . 31
2.3 Literature review . . . 32
2.3.1 Resource allocation with fairness based on single server . . . 33
2.3.2 Resource allocation with fairness based on heterogeneous servers . 38
2.3.3 Resource allocation with fairness based on heterogeneous server and placement constraints . . . 40
2.3.4 Resource allocation with fairness with cost efficient approaches . . 42
2.4 Summary . . . 44

3 Methodology **45**

4 Proposed fair resource allocation algorithms **49**
4.1 Introduction . . . 49
4.2 MLF-DRS: A Multi-Level Fair Resource Allocation Algorithm in Cloud computing . . . 51
4.2.1 MLF-DRS allocation . . . 52
4.2.2 Queuing in MLF-DRS . . . 55
4.2.3 Numerical Evaluations . . . 57
4.2.3.1 Numerical evaluations with two users . . . 57
4.2.3.2 Numerical evaluations with four users . . . 59
4.2.3.3 Evaluation based on fairness properties . . . 61
4.3 A Fully-Fair Multi Resource Allocation Algorithm in Cloud Environments (FFMRA) . . . 64

4.3.1 Motivation . . . 64
4.3.2 FFMRA . . . 66
4.3.2.1 Problem formulation . . . 67
4.3.2.2 Example . . . 70
4.3.2.3 Fairness measure . . . 71
4.3.2.4 A scenario with two users . . . 72
4.3.2.5 A scenario with more than two users . . . 73
4.3.3 Fairness properties analysis . . . 75
4.4 H-FFMRA . . . 79
4.4.1 Motivation . . . 79
4.4.2 Fair allocation properties in a multi-server setting . . . 80
4.4.3 System setting . . . 81
4.4.4 H-FFMRA . . . 83
4.4.4.1 Problem formulation . . . 83
4.4.4.2 H-FFMRA allocation . . . 84
4.4.4.3 An example . . . 85
4.4.5 Fairness . . . 87
4.4.6 Fairness properties satisfied by H-FFMRA . . . 87
4.5 A New Approach to Calculate Resource Limits with Fairness in Cloud Environments . . . 93
4.5.1 Motivation . . . 93
4.5.2 Fairness in Kubernetes . . . 94
4.5.3 System Design . . . 98
4.5.3.1 Mathematical implementation in Kubernetes . . . 98
4.5.4 Practical example . . . 99
4.6 Summary . . . 101

5 MRFS: A Multi Resource Fair Scheduling Algorithm in Cloud Computing 102
5.1 Introduction . . . 102
5.2 motivation . . . 103
5.2.1 Basic resource scheduling setup . . . 104
5.2.2 MRFS . . . 105
5.3 Summary . . . 113

6 Applicability of the proposed algorithms 114
6.1 Introduction . . . 114
6.2 CloudSim . . . 114
6.2.1 CloudSim Architecture . . . 115
6.2.2 CloudSim classes . . . 117
6.2.3 Deploying proposed algorithms using CloudSim classes . . . 121
6.3 The applicability in BT's infrastructure . . . 122
6.3.1 Kubernetes . . . 122
6.3.2 Pod . . . 124
6.3.3 Deployment in Kubernetes . . . 126
6.3.4 API Server . . . 128
6.3.5 Scheduling in kubernetes: . . . 130

6.3.5.1 Affinity and anti-affinity . . . 131
6.3.5.2 Resource quota and namespaces . . . 132
6.3.6 Resource allocation . . . 132
6.3.7 Kubernetes services . . . 133
6.3.7.1 minikube . . . 135
6.4 Models for implementing proposed algorithms in Kubernetes . . . 136
6.5 Summary . . . 138

7 Evaluations and Results 140
7.1 Introduction . . . 140
7.1.1 Workload . . . 140
7.1.1.1 Workload Generation . . . 141
7.1.2 Vm configuration . . . 143
7.2 Evaluations: MLF-DRS . . . 145
7.2.1 Resource Allocation . . . 145
7.2.2 Resource utilization . . . 146
7.2.3 Fairness . . . 148
7.2.4 Scheduling Time . . . 149
7.2.4.1 A scenario with five users . . . 151
7.3 Evaluations: FFMRA . . . 153
7.3.1 Resource allocation . . . 154
7.3.2 Fairness . . . 160
7.3.2.1 Fairness for group of users in each specific resource type . 160
7.3.2.2 Fairness for each individual user in each group in each specific resource type . . . 161
7.3.3 Resource Utilization . . . 163
7.4 Evaluations: H-FFMRA . . . 164
7.4.1 Resource allocation . . . 164
7.4.2 Resource utilization . . . 165
7.4.3 Fairness . . . 167
7.4.3.1 $\beta - fairness$ . . . 168
7.4.3.2 Jain's fairness index . . . 170
7.5 Evaluations: MRFS . . . 172
7.5.1 Allocation . . . 174
7.5.2 Sharing incentive . . . 175
7.5.3 Pareto efficiency . . . 178
7.6 Summary . . . 179

8 Conclusions 181
8.1 Summary . . . 181
8.2 Contributions . . . 182
8.2.1 A mutli-level fair allocation algorithm in the cloud . . . 182
8.2.2 A fully fair multi-resource allocation algorithm in the cloud computing . . . 182
8.2.3 A Completely Fair Multi Resource Allocation Approach in Heterogeneous Servers . . . 183

8.2.4 MRFS: A Multi-Resource Fair Task Scheduling in the Cloud Computing Systems . . . 183
8.2.5 A new Approach to Calculate Resource Limits with Fairness in Kubhernetes . . . 184
8.2.6 Future work . . . 184

A MLF-DRS traces **186**

B FFMRA traces **193**

C HFFMRA traces **197**

D MRFS traces **200**

Bibliography **205**

List of Figures

1.1 A representation of the original contributions of thesis - the purple blocks refer to the original contributions . . . 17

2.1 An example of Max-Min, and weighted Max-Min fairness . . . 22
2.2 Cloud computing service layers . . . 26
2.3 Resource allocation and provisioning in the cloud computing . . . 31
2.4 The evolutionary diagram of fairness in computing systems . . . 33

3.1 Key phases of constructive methodology used in this thesis . . . 45

4.1 The architecture of MLF-DRS . . . 52
4.2 The fair-share function in which the allocation is determined based on the situation of the corresponding requested resource . . . 54
4.3 Dequeuing jobs from the main queue to different queues based on dominant resources . . . 56
4.4 The order of executing tasks according to MLF-DRS scheduler . . . 57
4.5 Comparing DRF with MLF-DRS . . . 58
4.6 Resource allocation process in MLF-DRS . . . 60
4.7 FFMRA architecture . . . 67
4.8 Comparing FFMRA with DRF and DRBF in allocating resources . . . 73
4.9 An example with two users submitting their tasks over two servers . . . 79
4.10 The allocation of resources in DRF over two servers . . . 80
4.11 The allocation under H-FFMRA which indicates that resources are evenly allocated over two servers . . . 80
4.12 Pod processing in Kubernetes . . . 95
4.13 Using fair allocation and scheduling algorithms in Kubenretes - the figure illustrates each pod has been assigned with a namespace . . . 96
4.14 Pod configuration . . . 96
4.15 New resource limits assigned to pods A and B, considering DRf mechanism 97
4.16 Allocated limits for pods using one cluster installed with Minikube - the output represents the actual resource limit assignment for pods considering cluster resources . . . 100
4.17 Resource limits, calculated for all pods based on DRF policy . . . 100

5.1 An example of scheduling scenarios . . . 104
5.2 Scheduling tasks in different servers . . . 105
5.3 MRFS scheduler structure . . . 107
5.4 The optimal utility over the original utility . . . 110

5.5 The curve represents Pareto-efficiency where $\Pi_i^{'}$ represents the improved Pareto-efficiency . . . 110

6.1 CloudSim architecture (Calheiros et al. 2009; Vahora and Patel 2015) . . 115
6.2 A block definition diagram that illustrates the technical implementation of MRFS algorithm and its applicability in the cloudsim framework. . . . 120
6.3 A low-level architecture of designing and implementing resource allocation algorithms in the CloudSim . . . 121
6.4 Kubernetes architecture (Pscaer 2018) . . . 122
6.5 A sample deployment using YAML . . . 126
6.6 Deployment using resource requests, and limits . . . 127
6.7 Kubernetes API server (Schimanski and Hausenblas 2018) . . . 128
6.8 A tree that indicates how information is used to model our approach . . . 129
6.9 Resource allocation cycle in Kubernetes . . . 133
6.10 Kubernetes services abstraction . . . 134
6.11 A high level abstraction of Kube-batch . . . 135
6.12 A block definition diagram that indicates how to select an appropriate policy based on the optimization goals . . . 137
6.13 A requirement diagram that indicates how to integrate scheduling and allocation mechanisms based on the system requirements . . . 138
6.14 A sequence diagram that indicates how different algorithms could be applied in Kubenretes framework . . . 139
6.15 A block definition diagram, representing the integration of proposed algorithms in Kubenretes framework . . . 139

7.1 Vertical stretch . . . 143
7.2 Sample workload generation based on requested resources, taking into account CPU, and memory . . . 144
7.3 Resource allocation in MLF-DRS . . . 146
7.4 CPU allocation in MLF-DRS . . . 147
7.5 RAM allocation in DRF and MLF-DRS . . . 147
7.6 The comparison of resource utilization in DRF and MLF-DRS . . . 148
7.7 Jain's fairness index to evaluate how resources are fairly shared among users in Users in DRF and MLF-DRS . . . 149
7.8 The effects of server capacity on equal allocation of resources to tasks with dominant resource on a particular resource type under MLF-DRS policy . . . 150
7.9 Dominant CPU and RAM for each task in 1000 iterations. The higher value means the corresponding task is dominated on CPU or RAM . . . 152
7.10 The response time with respect to the submitted tasks either dominant on CPU and RAM. . . . 153
7.11 The frequency of dominant resources . . . 154
7.12 The fraction of requested resources by the users over the capacity of server 156
7.13 Allocated CPU for all users in DRF and FFMRA . . . 156
7.14 The comparison of allocated resources under FFMRA policy . . . 157
7.15 Allocated RAM for all users in DRF and FFMRA . . . 158
7.16 Allocated Disk for all users in DRF and FFMRA . . . 159
7.17 Sharing resource pool capacity among users with dominant resources . . . 161

7.18 Sharing resource pool capacity among users with non-dominant resources 161
7.19 Fairness for users with dominant resources 162
7.20 Total allocated tasks under DRF and FFMRA 163
7.21 The comparison of resource utilization in DRF and FFMRA 164
7.22 Allocated CPU to all users 165
7.23 Allocated RAM to all users 166
7.24 Allocated resources for four users under H-FFMRA policy over 30 servers 166
7.25 resource utilization in H-FFMRA, and MHDRF 167
7.26 The fair distribution of resources among servers, considering dominant resources 168
7.27 The fair distribution of resources among servers, considering non-dominant resources 169
7.28 Beta fairness for dominant, and non-dominant shares 170
7.29 The Jain's index for users with dominant resources 171
7.30 The Jain's index for users with non-dominant resources 171
7.31 The number of tasks scheduled as dominant resources in three servers under Round Robin and MRFS policies 173
7.32 The proportion of entire resource pool for tasks with dominant and non-dominant resources in presence and absence of MRFS 173
7.33 The proportion of entire resource pool, allocated to tasks with dominant and non-dominant resources 174
7.34 Total allocated resources to users across servers with and without MRFS 174
7.35 Total allocated tasks for eight incoming demands under MRFS policy . . 177
7.36 The values for δ for submissions with dominant and non-dominant resources 177
7.37 The values for Λ and α in 1000 iterations 178
7.38 The index for Pareto-efficiency for three users over the servers with different resource capacities - The capacities are considered as total capacity including both CPU and RAM resources. 179

List of Tables

2.1 The summary of multi-resource allocation mechanisms, emphasis on fairness properties such as Sharing Insentive (SI), Pareto Efficiency (PE), Strategy Proof (SP), and Envy Free (EF) as well as their main approaches. 43

4.1 Scheduling order based on the new queuing model. 57
4.2 Comparing allocation for DRF and MLF-DRS 58
4.3 Resource allocation in DRF and MLF-DRS approaches 59
4.4 The allocation of resources in DRF and FFMRA with resource capacity (9 CPU, 18 RAM) 66
4.5 The allocation and utilization of resources in FFMRA where user 1 has dominant share in RAM and user 2 has dominant share in CPU. 71
4.6 The allocation and utilization of resources in DRF where user 1 has dominant share in RAM and user 2 has dominant share in CPU. 71
4.7 Resource allocation in three different algorithms with resource capacity (18 CPU, 36 RAM) 74

7.1 Server configuration 145
7.2 The comparison of scheduling time in DRF and FFMRA 150
7.3 Server configuration for five users 154
7.4 159
7.5 The value of β in randomly selected iterations for tasks with dominant and non-dominant resources 176
7.6 Determining α derived by the value of δ from randomly selected iterations 176

Abbreviations

BAA	Bottleneck Aware Fairness
BBF	Bottleneck Base Fairness
CEEI	Competitive Equilibrium from Equal Incomes
DRF	Dominant Resource Fairness
FFMRA	Fuuly-Fair Multi Resource Allocation
FFU	Fair Share Unit
IaaS	Infrastructure as a Service
K8s	Kubernetes
MLF-DRS	Multi-Level Fair Resource Scheduling
MMV	Maximizing Minimal Value
PaaS	Platform as a Service
PE	Processing Unit
QoS	Quality of Service
RAS	Resource Allocation System
SaaS	Software as a Service
SI	Sharing Incentive
SP	Strategy Proof
TSF	Task Share Fair
VM	Virtual Machine
WMM	Weighted Max Min
YARN	Yet Another Resource Negotiator

Symbols

K	A set of heterogeneous servers
$\mathbf{s}$	Any server in a set k
ω	weights on resources
U	Users in the system
$J(x)$	Jain's fairness index
i	A user in a set U
R	The vector for different types of resources
X	Demand vector
r_i^k	requested resources by a user in vector X
k	Types of resource in R
Π_i^k	The allocation of a resource to a user
$\Psi(\Pi_i^k)$	The number of tasks allocated to a user
D	Dominant resources vector
$\tilde{D}$	Non-dominant resource vector
d	a dominant resource in D
$\tilde{d}$	a Non-dominant resource in $\tilde{D}$
f^k	The fair-share of a resource
C^k	The maximum capacity of a resource type
t	time
Q	A vector for queues
q	A queue in the system
GAR	Global Aggregate Resource
ρ	The proportion of the resource pool
φ	The proportion of ρ in each server s
β	Beta-fairness index

G	Groups of users
ϑ	The distribution of resources in each users's group
γ	The distribution of resources based on the proportion of total capacity of all servers
A	Available resources
δ	consumption factor
Δ	group of consumption factors in each server
α	Penalty variable
ζ	Average utilization of all servers
μ, λ	Lagrangian multipliers
ω	resource weight
Λ	Lambda variable
dp	dominant pod
P	a vector for pods
LP	resource limits for a pod
xp	resource requests for a pod
S_{LP_r}	Aggregate resource limits for a pod
f_{rp}	weight on a resource for a pod
Al	Allocatable resource for a node

Chapter 1

Introduction

1.1 Fairness in computer science

The fair allocation of resources is a significant milestone in computer science as any type of resource should be distributed among users with diverse requirements (Baruah et al. 1993). The fairness problem is widely considered in communication systems where it plays an important role in dividing the bandwidth between the links. Despite the rapid growth of network capacity, the traffic is also increased dramatically which contributes to resource scarcity in some points. Due to this drawback, users may not be able to receive a desirable share based on their demands. The Unfair allocation may occur without considering suitable sharing policies. The subsequent of such allocation leads to resource wastage, scarcity, starvation, and excessive allocation (Ahmed et al. 2009). However, the problem of fairness is not limited to the environments such as computer networks where a single type of resource is taken into account. In particular, resource allocation with fairness is considered in cloud computing systems with heterogeneous resources and servers. Therefore, in this thesis, we investigate the existing problems regarding fairness in cloud computing systems concerning the desirable fairness properties.

1.2 Fairness properties

Every fair resource allocation algorithm is subjected to meet some desirable fairness properties. In particular, an algorithm captures a perfect allocation if it satisfies all

important fairness criteria such as *sharing-incentive*, *strategy-proof*, *Pareto-efficiency*, and *envy-free*.

1.2.1 Envy-free

Based on the envy-free property, an allocation is envy-free if no user complains others' allocations or there is no user who prefers an allocation of others over a set of resources that he/she receives (Wang et al. 2013). In order to judge a true measure of the fairness, it is necessary to determine the utility functions of users to know whether they behave envy against others (Rajaram 2014). Hence, assuming that there are two users $i, j \in U = (1, 2, ..., n)$ where U denotes the number of users in the system. Accordingly, the allocation Π is envy free if the number of tasks allocated to user i satisfies $\Psi(\Pi_i) \geq \Psi(\Pi_j)$, where Ψ indicates the number of tasks allocated to users i and j.

1.2.2 Pareto-efficiency

An allocation meets the Pareto-efficiency if it grants an allocation Π for all possible allocations Π' as $\Psi'_i(\Pi_i) > \Psi_i(\Pi_i)$ (Poullie et al. 2018). In other words, under the Pareto-efficiency, no allocation is highly efficient than the current allocation. The Pareto efficiency does not obtain the fairness, however, it is still a favorable property in all conditions. The allocation of a set of resources R at time t is referred to Pareto-efficient if in each time t, the fraction of each specific resource r is allocated to users since, no user can better-off his/her allocation by worsening-off others' allocations (Psomas 2014). It has been discussed that under a perfectly complementary utility function, any non-wasteful allocation is Pareto-efficient when any kind of unallocated resource is assigned to users (Poullie et al. 2018). In this work the authors have highlighted the correlation between the fairness and Pareto-efficiency. There is also a relationship between Pareto-efficiency and proportional fairness. Starting with a contradict case, it has been assumed that a proportionally fair allocation, is not Pareto-efficient. In this case, there is an allocation as each user is given by at least an equal utility and also there is at least a user with a higher utility compared to others. Let's assume n users, arriving at l levels, taking into account that they do not prefer to relinquish their allocated resources. In such a condition, Pareto-efficiency is not achievable. Due to this, the most proportion of resources that have already been allocated to current

users could be envied by those users who have arrived during next intervals. Overall, the non-wasteful allocation is an adequate condition for Pareto-efficiency considering the concept of perfectly complimentary utilities (Avital and Noam 2012).

1.2.3 Strategy Proof

Despite other fairness properties, it is very challenging to achieve the Strategy-Proof, and in most cases, it is not possible at all (Li and Xue 2013). An allocation is Strategy-Proof if a user is unable to maximize his/her allocation by misreporting demands. Consequently, a user may stimulate others to report their utility functions. Assume that D denotes the demands vector of a user i on resources R and Π_i as the allocation of user i, reporting demands D_i truthfully. Also, there is a user who misreports demands, indicated by D'_i as $D_i \neq D'_i$. Therefore, according to the strategy-proof, $\Psi_i(\Pi_i) \geq \Psi_i(\Pi'_i)$. Theoretically, when fairness is still maintained, it is not necessary to satisfy in practical scenarios. It is even impossible for a system to determine which user has truthful submissions or which one intends to manipulate the system. This is due to that there is not enough information that a user may use to misreport his/her demands. Based on the number of users or how big a system is, strategy-proof can be maintained such as a case mentioned in (Zahedi and Lee 2014*a*) where it exists, considering a large number of users.

1.2.4 Sharing-incentive

The sharing-incentives property (Zahedi and Lee 2014*b*) guarantees that no user can better-off others when resources are regularly shared (Poullie et al. 2018). In terms of equal distribution of resources, each type of resource should be equally divided among users. Otherwise, if users have been given weight, then all the resources must be proportionally allocated to them. Therefore, an allocation meets the sharing-incentive if every user i betters off others' utilities U under an equal resource division. Meeting the SI in data centers where multi-resource allocation is considered, is a bit challenging and it is a different notion compared to economics perspectives (Poullie et al. 2018). Nonetheless, in cloud data centers, where resources are distributed among different servers, the sharing-incentive must guarantee that each user i gets at least a fair division of resources. In summary, if there are n users in the server and different types of resources are available with a specific capacity, then each user should receive at least $1/n$ of resources

(Zhao, Du, Lei, Chen and Yang 2018). On this occasion, it is possible to guarantee that sharing-incentive has been satisfied.

Apart from the above-mentioned features, there are also other fairness criteria as follows:

1. *Single resource fairness*: If there is only one type of resource in the resource pool (Lin and Su 2017). In this case, the allocation could be relaxed to Max-Min fairness.

2. *Bottleneck fairness*: If there are multiple types of resources and also there are many users, if all users share the same type of resource, then the allocation could be relaxed to the *Max-Min* fairness (Wang et al. 2013).

3. *Population monotonicity*: In a real-time environment where a user leaves the system in a specific period of time and gives up all the resources, then other users in the system should not experience any reduction in allocated resources (Ghodsi et al. 2011).

1.3 Efficiency and non-wasteful allocation

Efficiency is one of the important aspects of resource allocation with fairness in computing systems. The basic intuitive notion of efficiency is the maximum allocation of any resource type to users without wasting it under the notion of Pareto-efficiency (Avital and Noam 2012). It has been already confirmed that in perfect complimentary utilities, the efficiency could be interpreted as the Pareto-efficiency. Accordingly, an allocation is subjected to be non-wasteful if for each user i, all amount of resource R is advantageous for the utility of each user which is indicated by $U(x_i)$. Accordingly, a user may get more resources by improving the utility of a specific resource type. An allocation Π is called wasteful if for users i and j, a resource type is allocated for the value of 0 to user i and more than 0 to user j (Shah 2017).

1.4 Fairness in cloud computing

Resource allocation with fairness has been widely considered as one of the challenging issues over the last two decades. Initially, fairness in computer science has been investigated in computer networks by proposing different mechanisms using generally accepted algorithms such as Max-Min and proportional fairness. (Bonald et al. 2006). However, with the rapid expansion of modern computing systems, the fairness issue has received much attention. Despite other distributed systems, cloud computing is particularly recognized in the heterogeneity of resources and servers (Lopez-Pires and Baran 2017). In other words, a data center in the cloud is likely to be established by different servers, including diverse configurations in terms of resources, e.g., CPU, memory, bandwidth, and disk storage (Wang et al. 2013). Correspondingly, users pose a significant interest to submit jobs, consisting of multiple resource types. Due to such diversity, the recent investigations have indicated that almost over 50% of resources in the cloud are wasted due to the resource fragmentation and inefficient utilization of resources (Nehru et al. 2016).

The definition of fairness in the context of cloud computing is very complex compared to the traditional computing frameworks where the efficiency in resource allocation has been determined, using,e.g., the makespan (Poullie et al. 2018). From a single resource perspective, it is straightforward to equally divide a particular resource among a certain number of users. However, in a multi-resource environment like cloud computing, the equal resource division is hard to define. This is due to that the collection of resources in the cloud are not comparable (Poullie et al. 2018). In particular, some tasks submitted by users may include heavy requests on a specific resource type such as mathematical operations that require a considerable amount of CPU. Nonetheless, others may request a significant ratio of bandwidth to transfer large files through the web. Furthermore, the dependency among these resources is still challenging as all types of them should be considered in resource allocation decisions. Therefore, users must be allocated with different ratios of resources, while these sets of resources are not comparable.

To overcome the fairness problem in cloud computing, and as the first attempt, the Dominant Resource Fairness (DRF)(Ghodsi et al. 2011) has been proposed. The dominant resource has been defined as the highest demand in which other demands are indicated as fractions of the maximum resource capacity. DRF has rapidly attracted much attention

as it has achieved several desirable fairness features such as sharing-incentive, resource monotonicity, envy-free, Pareto-efficiency, and strategy-proof (Ghodsi et al. 2011). It has been also implemented as a scheduler on top of well-known platforms such as YARN (Vavilapalli et al. 2013) and Apache Mesos (Hindman et al. 2011). However, under the DRF policy, at least one type of resource is not fully utilized. It is even worse if DRF is deployed separately in multiple servers. The allocation under this criteria has been expressed as a highly inefficient allocation (Wang et al. 2013). In other words, DRF has tried to maintain fairness among users with different dominant resources. However, it has failed to achieve the efficiency and full utilization of resources. Dozens of extensions have introduced different approaches to address the existing issues associated with DRF (Wang et al. 2013; Wang et al. 2016; Khamse-Ashari et al. 2017). We will indicate that these approaches have failed to meet some important fairness features such as sharing-incentive and Pareto-efficiency for a certain number of users that may result in making an allocation intuitively unfair.

Besides, the task scheduling with fairness in cloud computing has become a significant issue due to the heterogeneity of resources and servers (Farias et al. 2020). Indeed, the responsibility of a scheduler is to map incoming tasks/jobs to the most suitable servers/hosts to enable users to receive the best Quality of Service (QoS), e.g., response time and maximum resource utilization. In terms of task scheduling, DRF has employed the progressive-filling algorithm since users' tasks are scheduled based on the minimum dominant resource. While the scheduling in DRF is limited to the single server, Dominant Resource Fairness in Heterogeneous servers (DRFH) (Wang et al. 2013) has introduced an alternative policy as such, each server hosts only one user's tasks. The scheduling, taking into account this criterion violates the sharing-incentive property as under the DRFH, some users are unable to receive at least 1/n of the maximum capacity of the resource pool. Hence, the understanding of finding an optimal server to host users' tasks, dominated by multiple resource types is a significant concern. A different approach in (Khamse-Ashari et al. 2017) has introduced a scheduling algorithm in presence of placement constraints in which users may only receive resources from a subset of servers. This scheduling mechanism has fallen short in satisfying the Pareto efficiency. Accordingly, this thesis tries solve these problems by proposing fair resource allocation and task scheduling algorithms.

1.5 Trade-off between the fairness and efficiency in cloud computing

It has been studied that considering only the fairness is not enough to maintain the satisfaction between users and providers (Bertsimas et al. 2012). Indeed, the resource allocation problem becomes more complex when the fairness and efficiency metrics are considered. The term efficiency in resource allocation is referred to the maximum/full utilization of a particular resource type. Hence, on the one hand, increasing resource utilization is desirable in terms of efficiency, and on the other hand, users must be satisfied with the number of resources they receive. Actually, failing to satisfy either the fairness or efficiency leads to a wasteful allocation (Xiao et al. 2013). Therefore, it is essential to seek an optimal trade-off strategy to provision resources so that a higher degree of fairness and efficiency is achieved.

For example, the Max-min fairness as a widely accepted fair allocation policy meets the Pareto-efficiency, while it does not maintain the maximum utilization of resources. This problem has been investigated in (Tang et al. 2004), suggesting how to deal with this significant drawback. Apart from the Max-Min fairness, the alpha variable in α-fairness is a tunable parameter that establishes a trade-off between the fairness and efficiency (Jin and Hayashi 2018). However, the α-fairness is not always a suitable candidate to guarantee such a trade-off. The details of overcoming this issue have been explored in (Sediq et al. 2012).

Maintaining fairness and efficiency could be straightforward in general resource allocation problems such as computer networks that are solely based on a single resource type. However, in heterogeneous environments like Cloud computing, it is very challenging to meet this trade-off. Despite the heterogeneity of resources in cloud computing, some frameworks such as the Hadoop slot scheduler have applied a simple approach in resource allocation decisions (Gautam et al. 2015; Wang et al. 2013). The allocation under this criteria has an adverse effect on the full utilization of resources which may result in a highly inefficient allocation. As it has been discussed in 1.4, under the DRF policy, all the resources in the data center have not been fully utilized. To address this fundamental issue, establishing a trade-off between fairness and efficiency is likely to be an optimal solution (Zahedi and Lee 2014*a*). To have a better understanding of the

problem, we illustrate a simple example, considering two users, each demands CPU and RAM to schedule tasks. Assume that user 1 requests for 2GB of RAM and 3 CPUs for each job. Also, user 2 requests for 2 GB of RAM and 1 CPU for each job. The system also has a capacity of 6 GB RAM and 4 CPUs. In this example, the concept of fairness could be the amount of allocated resources based on the proportion of both users' demands. According to this example, and based on the DRF allocation policy, 0.76 and 1.71 of jobs are allocated to users 1 and 2 respectively, with overall 2.47 allocated jobs. Hence, according to the requested resources by users, user 1 gets only 0.17 jobs, while user 2 receives 2.83 jobs which yields overall 3 jobs to both users.

As an initial attempt, the work in (Joe-Wong et al. 2012*a*) has proposed fairness functions in a unifying framework to achieve a trade-off between fairness and efficiency, taking into account the impact of fairness properties in achieving this trade-off. They have also investigated the application of α-fairness to achieve such a trade-off. Although tuning the α variable to a higher value may result in better fairness, it could not maintain the efficiency. The joint application of the α and proportional fairness has been further explored as another solution to achieve a trade-off subject to appropriately adjusting the α variable (Khamse-Ashari et al. 2017). Although this approach has achieved this trade-off, the Pareto-efficiency has not been satisfied. Correspondingly, in this thesis, we take into account a distinctive approach compared to the recent solutions. In particular, instead of using the α and global dominant share, the aggregate dominant and non-dominant resources are introduced to achieve the efficiency and fairness trade-off (see Chapter 4). Accordingly, the equalization of non-dominant resources is taken into account as it has been ignored in recent approaches. Indeed, we formulate a maximization problem, considering a correlation between dominant and non-dominant resources that guarantee this trade-off.

1.6 Intuitive fairness in cloud computing

In economics, the concept of intuitive fairness has been reviewed in two different aspects. In (Merkel and Lohse 2018) authors have studied whether fairness could be intuitive based on how people judge fairness and respond to it in different conditions. Indeed, this approach is out of the scope of this thesis. Therefore, we consider another perspective of exploring the notion of intuitive fairness considering fair allocation algorithms and

fairness properties. In particular, this is possible by considering the satisfaction of all individuals based on their preferences. We take a closer look at this problem in different scenarios, taking into account a traditional example of cutting problem (Babaioff et al. 2019). Based on this work, an example including two players has been considered. The one is the cutter who divides a good into two portions while another player selects based on what he/she prefers; in this case the allocation is intuitively fair. In particular, each player gets exactly based on what his/her preference is. This allocation is the basic concept of Max-Min Share (MMS) in competitive equilibrium (Aziz et al. 2019). In another scenario, we assume two users A and B, and there is a good called Y. It has been also assumed that both users have not any preference on that good. As a result, Y could be divided equally among both users so that each user gets $Y/2$ of that good. Hence, according to the sharing-incentive property, this allocation is again intuitively fair. Also, this type of allocation has been further investigated under the envy-free property as one of the important fairness features (Kolm and See 2002).

Fair allocation mechanisms such as Max-Min and proportional fairness have been identified as appropriate solutions to divide resources among users when they have different preferences on a specific item/resource. Generally speaking, the allocation under these algorithms is intuitively fair (Poullie and Stiller 2016) as users have no specific weight on a demand and they get an equal share of a resource based on their preferences. However, in the weighted proportional and Max-Min fairness, the allocations have been determined by weights assigned to requested resources (Srikant 2004). Consequently, judging whether an allocation is intuitively fair may depend on the value of α where $1 < \alpha < \infty$. Accordingly, by increasing the value of α, the allocation becomes intuitively fair as it almost satisfies all users' preferences. Correspondingly, the low value of α may violate the Pareto-efficiency as increasing a user's allocation could decrease others' allocations. In this case, the allocation is not intuitively fair. Regardless of this, it is straightforward to achieve an intuitively fair allocation in the settings solely based on a single resource type (Poullie and Stiller 2016).

Therefore, we investigate the notion of intuitive-fairness in cloud computing with multiple types of resources. In section 1.4, we discussed DRF as a well-known policy for allocating resources with fairness in cloud computing. Unfortunately, DRF and its recent developments have ignored the notion of intuitive fairness. This significant issue has been investigated in (Poullie et al. 2018), that not satisfying all fairness properties

leads to an intuitively unfair allocation. Accordingly, the authors have explored the concepts of L_1 and L_∞ norms (Li et al. 2016) in DRF and Asset fairness to determine overall resource utilization. The L_1 norm is a measure to quantify the overall allocated resources to users in which their demands are equally weighted. Asset fairness generalizes max-min fairness by employing the L_1 norm. Hence, L_∞ could be used rather than L_1 when multiple types of resources are considered. Therefore, L_∞ specifies the value of a requested resource considering its dominant share, which is the highest demand related to the overall allocatable amount of a resource type. It is still arguable to tell which norm is fairer, as under DRF policy a user with a dominant resource type k gets a small fraction of that resource. While others with a dominant resource on a different resource type get considerably more quota. To have a better understanding of the problem, we refer to an example in (Poullie et al. 2018). It is assumed that there is a system with two resource types that are divisible with a capacity vector of $(1, 1)$. Also there are three users A, B, and C by demand vectors $(1, 0)$, $(0, 1)$, and $(1, 1)$, respectively. User A requests only the first resource type, and user B requests the second one, while user C requests both resources. Consequently, under the DRF policy, users A and B get a 0.5 fraction of resources, while user C receives the same fraction of 0.5. As a result, this allocation is not intuitively fair as this user has intensive demands on both resources. It is worth mentioning that DRF is not intuitively fair concerning the Leontief functions (Poullie and Stiller 2016) in which the excessive allocation is allocated among users.

Generally speaking, under the allocation in the above-mentioned example, the Pareto-efficiency has not been satisfied as increasing the allocation for users A and B, has decreased the allocation for User C. To be intuitively fair, the allocation vectors for users A, B, and C must be $(0.25, 0)$, $(0, 0.25)$, and $(0.75, 0.75)$ respectively. To the best of our knowledge, the Pareto-efficiency in recent approaches has not been satisfied for some users with tasks dominated on a particular resource type. This problem seems to be significant in large-scale scenarios in which there is a large number of demands with a remarkable diversity in computational resources. Therefore, we believe that the resource allocation and scheduling mechanisms are subjected to satisfy Pareto-efficiency as well as sharing-incentive to achieve the notion of intuitive fairness. Hence, to achieve this important criteria, this thesis tries to propose different resource allocation and task scheduling algorithms to satisfy desirable fairness properties, specifically, the sharing-incentive and Pareto-efficiency for all users.

1.7 Research problems

We already discussed that all the allocation and scheduling mechanisms are required to satisfy desirable fairness properties for all users to achieve an intuitively fair allocation. Accordingly, the recent approaches suffer from the following issues:

1.7.1 Not consider the fair-share boundaries in resource allocation decisions

The fair-share has been considered as a significant criterion to fulfill the sharing-incentive property. The equalization of dominant resources based on the fair-share has been explored in (Zhao, Du and Chen 2018). However, the numerical evaluations, conducted in 4.2.3 has revealed that the equalization process solely based on dominant resources may not satisfy Pareto-efficiency and sharing incentive in large-scale scenarios. This issue may establish an obstacle against utility maximization for tasks, dominated by different resource types. In recent developments, there is no specific approach to consider the boundaries of fair-share in resource allocation decisions (see 4.2 for more details).

1.7.2 Not employ the queuing isolation in scheduling time

As it has been already mentioned in 1.7, modern data centers are recognized in the heterogeneity of resources and servers. There is no specific study around fairness in queuing mechanisms based on dominant resources. We believe that this problem becomes significant when the front of a queue is populated by tasks, dominated by a certain resource type. In this case, other tasks with a different type of dominant resource are subjected to wait for a long time until being scheduled (more details are provided in section 4.2.2).

1.7.3 Not consider fully fair resource allocation, among tasks with different resource types

In dominant resource-based allocation policies, both in single and multi-server approaches, it has not been investigated whether an accurate proportion of the entire resource pool is allocated to tasks with different types of dominant resources. Consequently, some tasks dominated on a particular resource type may occupy a more

proportion of the resource pool. We will show that this leads to the starvation issue, as some users are unable to maximize their allocations with nonidentical dominant resources. Accordingly, such an allocation may adversely violate the Pareto-efficiency feature. Moreover, there is no certain strategy to measure the fair distribution of resources among users with respect to heterogeneous resources and servers.

1.7.4 Not consider the number of dominant resources in multi resource scheduling in cloud computing

We believe that populating a server with an uneven number of diverse dominant resource types may increase the competition among tasks with identical dominant resources. Consequently, the Pareto-efficiency and Sharing incentive features could not be achieved in large-scale scenarios. Hence, it is crucial to re-think the task scheduling by keeping the number of dominant resources in an equal condition concerning each server.

1.7.5 Not capture the fairness in Kubernetes

The Kubernetes is a cloud-native solution that has been introduced to orchestrate containers in a fully managed manner. The scheduler in Kubernetes is responsible for finding the most suitable nodes to place pods. Since the scheduler does not consider resource limits in scheduling time, many pod evictions may occur due to this important drawback. Additionally, as users may request many replicas of pods, there is no fair mechanism to manage those pods and assign resource limits in a way that all users benefit from the best QoS with a minimum number of evictions.

1.8 Research questions

This thesis tries to answer the Research Questions (RQs) based on the problems discussed in section 1.7 as follows:What is the intuitive fairness in cloud computing?. Accordingly, what is the impact of violating desirable fairness properties, such as the sharing-incentive and Pareto-efficiency features to achieve an intuitively fair resource allocation? What is the relationship between the fair-share and resource allocation decisions? Does the queuing isolation for incoming tasks,

considering different types of dominant resources help to reduce the response time for a certain number of tasks? Is it possible to achieve a fairer allocation, taking into account equalizing dominant and non-dominant resources? How to achieve a fully fair resource allocation, considering the trade-off between fairness and efficiency while maintaining a correlation between dominant and non-dominant resources?. To achieve this, how to fairly share resources among groups of users with dominant and non-dominant resources? As Jain's index is a specific metric to evaluate the fair allocation of resources among users, is there any mechanism to evaluate the fair distribution of resources in a multi-resource environment, among a group of users with dominant and non-dominant based demands? Is there any relationship between the number of tasks dominated on a specific resource type with Pareto-efficiency and sharing incentive properties concerning each server? If it is, does equal numbers of non-identical dominant resources in each server could achieve and improve these properties? What is the main reason of occurring pod evictions in the Kubernetes framework? Does applying fair resource allocation algorithms solve this problem?

1.9 Aims and objectives

The main aim of this thesis is to achieve an intuitively fair resource allocation in cloud computing so that the fairness properties are achieved for all users either in the single and multi servers perspectives. To reach this aim, we try to answer the research questions in 1.8. Hence, the general objectives of this research are listed as follows:

1.9.1 Studying resource allocation with fairness in cloud computing

Resource management in cloud computing has become of paramount importance during the last decade. In the meantime, fairness in resource allocation has established new research challenges in this area. Accordingly, the aim of studying fairness is to have a deep understanding of the existing problems associated with recent fair resource allocation mechanisms. This study tries to achieve the following objectives:

1. Understanding the notion of fairness in computer science and cloud computing systems.

2. Figuring out the main intuition behind fairness features such as sharing-incentive, Pareto-efficiency, strategy-proof, and envy-free.

3. Having a depth understanding of resource allocation mechanisms in cloud computing from single to heterogeneous servers settings.

4. Exploring the concept of intuitive fairness in cloud computing and the influence of satisfying sharing-incentive and Pareto-efficiency for all users on achieving it.

1.9.2 Investigating the existing problems associated with Dominant Resource Fairness(DRF) and its recent developments

The recent developments around Dominant Resource Fairness (DRF) have achieved several desirable fairness properties. However, recent investigations have revealed that they are not intuitively fair. Moreover, in large-scale scenarios, some users are unable to maximize their utilities. Accordingly, in this section the following objectives have been expected to be achieved:

1. Developing new fair allocation mechanisms in the single and multi-server points of view to achieve the notion of intuitive fairness.

2. Introducing a novel fair resource allocation mechanism based on the fair-share function, considering the boundaries of demand profiles.

3. Introducing a novel queuing mechanism to isolate various dominant resource types to prioritize them in scheduling time.

4. Equalizing both dominant and non-dominant resources to achieve a fairer resource allocation.

5. Defining new metrics to measure the fairness in cloud computing.

6. Achieving a trade-off between the fairness and efficiency by maintaining a correlation between dominant and non-dominant resources, through introducing the aggregate and global aggregate dominant resources.

1.9.3 Exploring the fairness problem in Kubernetes

The scheduler component in the Kubernetes framework fails to satisfy the fairness, as it does not consider the resource limits in scheduling time. This fundamental concern results in pod evictions and degrading QoS. Hence, the main purpose of this section is to satisfy the following objectives:

1. Modeling the fairness in the Kubernetes and integrate proposed algorithms to improve the functionality of the kube-scheduler.
2. Minimizing the possibilities of pod evictions in scheduling time.

1.9.4 Assigning incoming tasks fairly among servers with heterogeneous settings in cloud computing

.

The task scheduling in heterogeneous cloud environments has not considered the population of different types of dominant resources in each server. Failing to satisfy this important criterion contributes in violating the Pareto-efficiency and sharing incentive properties that may result in intuitively unfair resource allocation. Therefore, the following objectives are expected to be satisfied:

1. Defining new variables to maintain an equal distribution of tasks among servers with respect to dominant resources.
2. Achieving high degree of Pareto-efficiency and sharing incentive features.
3. Applying the Lagrangian multipliers to define the main maximization problem, aiming to find the most efficient servers to host incoming tasks.

1.10 Research methodology

In this thesis, we use the constructive methodology which introduces different steps that are required to conduct this research. Accordingly, we first perform a comprehensive literature review to figure out the existing problems associated with resource allocation

with fairness in cloud computing. Then, we propose different approaches to address those issues. We then investigate the applicability of those approaches in the modeling perspective and industrial context. Finally, we evaluate the proposed algorithms in the simulation environment. The details for the methodology are clearly discussed in chapter 3.

1.11 Original contributions

The main contributions of this thesis are categorized as follows:

1. **A multi level fair resource allocation algorithm in cloud computing (MLF-DRS)**: Proposing a new fair resource allocation algorithm, considering Max-Min fairness and proportionality. Accordingly, a decision making mechanism is taken into account to allocate resources among users with respect to dominant resources.

2. **A completely fair resource allocation algorithm in cloud computing (FFMRA)**: Proposing a novel fair resource allocation algorithm, and a new fairness measurement metric to evaluate the even distribution of resources among users with dominant and non-dominant resources.

3. **A fair resource distribution and allocation algorithm in heterogeneous cloud (H-FFMRA)**: Extending the notion of FFMRA in multi-server perspective, considering all allocation principles in FFMRA.

4. **A new approach to calculate resource limits with fairness in Kubernetes**: Defining a new model to integrate the fairness in Kubernetes, taking into account dominant resources in scheduling pods within corresponding nodes.

5. **A novel fair task scheduling mechanism in heterogeneous cloud (MRFS)**: Proposing a new task scheduling algorithm, trying to keep the system in a desired state to making sure that each server is occupied by tasks with equal number of dominant resources on each particular resource type.

Figure 1.1, illustrates the main contributions of this thesis. As can be seen in the figure, first of all we propose MLF-DRS (Section 4.2) as an inspiration of DRF, Max-Min

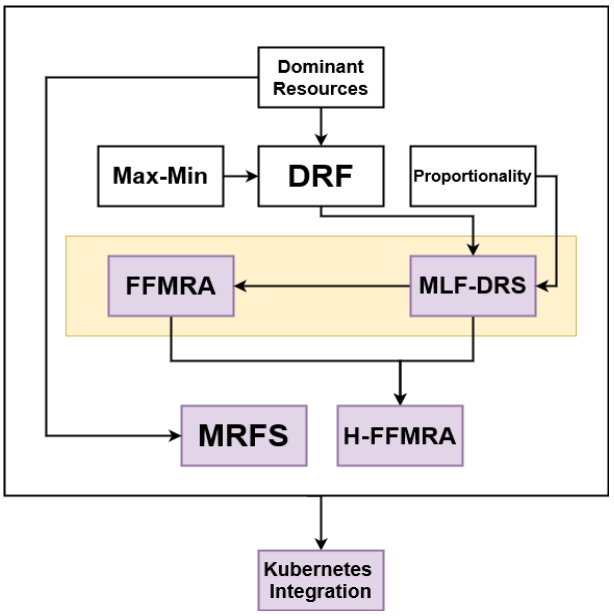


FIGURE 1.1: A representation of the original contributions of thesis - the purple blocks refer to the original contributions

fairness, and proportionality. Then FFMRA (Section 4.3) is introduced which aims to distributes resources fairly among users. FFMRA employs the allocation principles in MLF-DRS. While these algorithms are categorized within single server environments, MRFS (Section 5.2.2) and H-FFMRA (Section 4.4.4) are mainly proposed in multi-server profile. Finally, we propose a model to integrated all proposed algorithms in Kubernetes (Section 4.5, Chapter 4).

1.12 Publications

1. Hamzeh, Hamed Hemmati, Mahdi Shirmohammadi, Shervin. (2017). Bandwidth Allocation with Fairness in Multipath Networks. International Journal of Computer and Communication Engineering. 6. 151-160.

2. H. Hamzeh, M. Hemmati and S. Shirmohammadi, "Priced-Based Fair Bandwidth Allocation for Networked Multimedia," 2017 IEEE International Symposium on Multimedia (ISM), Taichung, 2017, pp. 19-24.

3. Hamzeh, Hamed Meacham, Sofia Botond, Virginas Phalp, Keith. (2018). Taxonomy of Autonomic Cloud Computing. International Journal of Computer and Communication Engineering. 7.

4. Hamzeh, Hamed Meacham, Sofia Virginas, Botond Khan, Kashaf Phalp, Keith. (2019). MLF-DRS: A Multi-level Fair Resource Allocation Algorithm in Heterogeneous Cloud Computing Systems. 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS).

5. Georgia, Isaac Meacham, Sofia Hamzeh, Hamed Stefanidis, Angelos Phalp, Keith. (2018). An Adaptive E-Commerce Application using Web Framework Technology and Machine Learning.

6. Hamzeh, Hamed Meacham, Sofia Khan, Kashaf Phalp, Keith Stefanidis, Angelos. (2019). FFMRA: A Fully Fair Multi-Resource Allocation Algorithm in Cloud Environments. IEEE SmartWorld, Ubiquitous Intelligence Computing.

7. Hamzeh, Hamed Meacham, Sofia Khan, Kashaf. (2019). A New Approach to Calculate Resource Limits with Fairness in Kubernetes. : IEEE International Conference on Digital Data Processing (DDP).

8. Hamzeh, Hamed Meacham, Sofia Khan, Kashaf Phalp, Keith Stefanidis, Angelos. (2020). MRFS: A New Multi Resource Fair Task Scheduling in Heterogeneous Cloud Computing (Accepted in 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)).

9. Hamzeh, Hamed Meacham, Sofia Khan, Kashaf Phalp, Keith Stefanidis, Angelos. H-FFMRA: A Multi Resource Fully Fair Resources Allocation Algorithm in Heterogeneous Cloud Computing. (in process - Springer Journal of Computing).

1.13 Thesis organization

This theses is organised as follows. In chapter 2, we review the basic and well-known fair resource allocation algorithms such as Max-min, and proportional. Then, we explore the basic concepts regarding the cloud computing and its service models. In the final section of the chapter, we conduct a comprehensive literature review on multi-resource allocation mechanisms in cloud computing. The methodology is discussed in chapter 3.

Chapter 4 discusses MLF-DRS, FFMRA, and H-FFMRA as our three proposed fair resource allocation algorithms as well as the model to calculate resource limits with fairness in Kuberentes framework.

In chapter 5, we introduce a fair new task scheduling mechanism called MRFS.

Chapter 6, investigates the applicability of proposed algorithms in the CloudSim and Kubernetes. This chapter primarily focuses on the technical background of the simulation environments and the models that describe their main functionalities in designing proposed algorithms.

In chapter 7, we evaluate the performance of proposed algorithms, taking into account different metrics such as resource allocation, utilization, and fairness. The experiments are conducted in the cloudsim simulation framework, driven by the randomly-generated workloads and Google workload traces. Finally, in chapter 8 we summarize the whole thesis in different sections along with future directions.

Chapter 2

Background and literature review

2.1 Introduction

The fair division of resources has been considered as one of the significant issues in different fields such as economics, computer science, and property management (Brams and Weber 1996). This is the knowledge of distributing a set of resources between a certain number of individuals based on their entitlements. Any fair division mechanism is subjected to meet some important requirements to ensure that each individual receives at least a fair share of any specific item. The first one is proportionality (Baruah et al. 1993), which means that every individual receives at least 1/n of his/her utility. The second one is equitability as each individual expects to get the same amount of resources based on his/her assessment, exactly the same as others' preferences. The last one is the Pareto-efficiency (Hansson 2004) as no individual can better off others' allocations. In computer science, these concepts are determined based on utility functions. For example, in the well-known fair resource allocation mechanisms such as Max-Min fairness (Jin et al. 2009), the utility of a user is maximized to satisfy at least one of the fairness properties, that have been already discussed in 1.2. In particular, the Max-Min and other algorithms such as proportional and α-fairness are generally used for environments where a single type of resource is taken place. Nonetheless, by the emergence of cloud computing technology, considering only a single resource type leads to an unfair and inefficient allocation (Wang et al. 2013). Consequently, the application of all these algorithms has been generalized in different approaches to fulfill all desirable fairness features. Accordingly, in this chapter, we will explore whether these algorithms have

been applied in cloud computing environments. Moreover, we will discuss how the fairness properties have been satisfied under these criteria. This chapter is categorised as follows. The section 2.2 is dedicated to the background study of the main concepts behind the basic fair resource allocation algorithms. Based on this, we discuss the cloud computing as well as its delivery and deployment models in 2.2.6. Moreover, the problem of fairness is investigated in each setup. We also explore the existing problems regarding the fair resource allocation and scheduling in cloud computing systems in 2.2.7. In the last section of this chapter, a systematic literature review is conducted in 2.3, considering different fairness approaches in cloud computing.

2.2 Background

2.2.1 Max-Min fairness

The feasible allocation of resources r to a set of n users is Max-Min fair (Bertsimas et al. 2011) if for each user i, the allocation Π cannot be increased without decreasing another user j' allocation indicated by Π_j in which $\Pi_j \leq \Pi_i$ and $i \neq j$. In other words, Max-Min fairness is possible if a user cannot increase his/her allocation without decreasing others' allocations. Under the Max-Min policy, the allocation of the smallest value is always increased. This procedure continues until all links are saturated. Accordingly, there is a Maximizing Minimal Value (MMV) problem as an optimization technique as follows:

$$maximize(min(\Pi_i))$$

Where Π refers to all possible allocations to users. Fundamentally, Max-Min fairness attempts to increase the allocation of a user, considering the least amount of a particular resource type. The Max-Min fairness has been widely used and investigated in different disciplines such as network resource sharing, routing, flow, and congestion control (Nace and Pioro 2006). There is also min-max fairness where the allocation of a user cannot be decreased without increasing another user's allocation. Max-Min fairness is also referred to the bottleneck optimality as it tries to divide the resources of a saturated link(fully utilized) fairly among users (Coluccia et al. 2012).

The max-min fairness algorithm works as follows. In the first step, the value of the fair-share is assigned to unsatisfied links. In the next step, the over-assignment from satisfied links are shared fairly among unsatisfied ones. In this thesis, we re-think the concept of Max-Min fairness to propose two new fair allocation algorithms in cloud computing. Similar to other works, the Max-Min fairness in our proposed approaches is fitted into the domain of multi-resource frameworks instead of the single-resource setting.

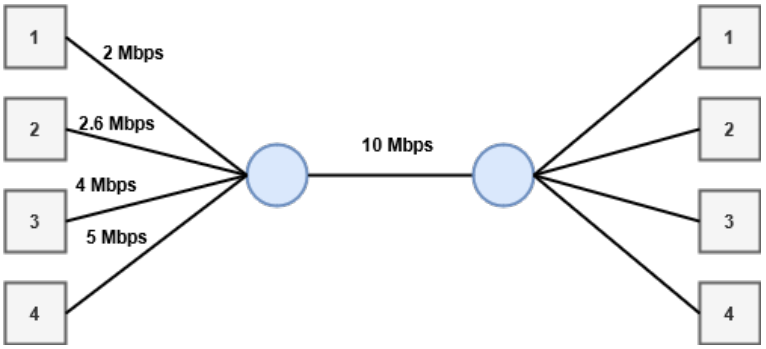


FIGURE 2.1: An example of Max-Min, and weighted Max-Min fairness

We already discussed in 1.6, the original use case of max-min fairness without considering the weights of preferences on a particular resource type is intuitively fair.

Example

Based on the example (Cheung n.d.) in Figure 2.1, there are four users with different demands, competing over a bottleneck link with the capacity of 10 Megabits per second (Mbps) . According to the Max-Min fairness, the fair-share of the specified capacity is determined for each link. In this example, 2.5 Mbps is specified for each user. At the first level, users 1-4 get 2,2.5,2.5,2.5 Mbps. Since, the demand of user 1 is 2, it is not possible to get the fair-share. Therefore, there is 0.5 Mbps over-assignment or unused bandwidth. In the next stage, the Max-Min fairness computes the fair-share of the unused resource of 0.1666 for users 2,3 and 4. Consequently, they receive 2.6 Mbps, 2.66666 and 2.66666 respectively. As, the user 2 in the second level has received 2.5 Mbps, there is still 0.0666 over-assignment. Accordingly, this value is divided among users 3 and 4. Consequently, the final allocation could be determined as follows. User 1: 2 Mbps, User 2: 2.6 Mbps, User 3: 2.7 Mbps and User 4: 2.7 Mbps. Based on the above allocations, user 1 is the first one to be maximized, considering the lowest demand, while others are maximized in sequence just after user 1.

2.2.2 Weighted Max-Min fairness

In real-world scenarios, users may have weight on a specific resource type. Accordingly, a user with a higher weight should be allocated more compared to others (Allalouf and Shavitt 2004). Therefore, the priority of users is determined based on the weights on each particular resource type. The resource allocation in the Weighted Max-Min Fairness (WMMF) is determined in different levels. First of all, based on the normalized weights, the allocation is calculated depending on users' demands. As a result, if two users have exactly the same weights, the WMMF initially satisfies a user with minimum resource demand. In the next step, only those users with unsatisfied demands are satisfied using the over-allocated amount of resources based on their weights.

Given the weights indicated by (w_i) and (w_j) for users i and j respectively, it is possible to say that the allocation is WMMF if it is not possible to increase Π_i without decreasing Π_j, where $\Pi_j/w_j \leq \Pi_i/w_i$, $(i \neq j)$.

Example

Taking into account the example in Figure 2.1, it is assumed that each flow has been assigned to a weight in which $w_1 = 1$, $w_2 = 2$, $w_3 = 1$, and $w_4 = 1$ that are the weights associated with flows 1 to 4 respectively. Hence, under WMM, all allocations are generalized by adding weights to them. First of all, WMM calculates the Fair Share Unit (FSU) using the following formulation:

$$FSU = unallocated/totalweight \tag{2.1}$$

Secondly, it allocates FSU to every weight unit. Finally, it calculates over-assignment values for all flows. Based on the example, the final allocations are as follows. Flow 1= 2.33333, flow 2= 3, flow 3= 2.33333 and flow 4=2.33333.

2.2.3 Proportional fairness

Considering a system with the resource capacity C and an allocation vector $\Pi = (\Pi_1, \Pi_2, ..., \Pi_n)$, considering n users. The allocation is proportional fairness if it is feasible (Allalouf and Shavitt 2004; Li et al. 2008):

$$\forall i, \Pi_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{n} \Pi \leq C \tag{2.2}$$

Using proportional fairness, it is possible to measure the multi-resource allocation, as it considers a user with different resource demands instead of a single resource type (Jalal et al. 2017). Proportional fairness could be relaxed to the max-min fairness in specific circumstances. WMMF and proportional fairness are intuitively fair based on the values of α. Hence, tuning this variable to an appropriate value could indicate whether these policies are intuitively fair or not. Since, the basic concept of proportional fairness is the best-fit approach in the multi-resource cloud that we also consider it in our proposed algorithms.

2.2.4 Alpha fairness

The fairness degree is specified using the alpha which is a tunable parameter to control the trade-off between fairness and efficiency (Jin and Hayashi 2018). The value of the alpha implies to specific conditions (Altman et al. 2008). If $\alpha > \infty$, then it denotes the MaxMin fairness which is the maximum fair allocation. If alpha equals 1, it refers to the proportional fairness and in a case such as 0, there is a maximization in throughput.

2.2.5 Quantitative fairness measure

To make sure that an allocation is fair, it is crucial to consider the quantitative measures to scale the system in terms of fairness. Quantitative approaches are being used in networking and computing to evaluate the fairness in resource allocation. There could be various quantitative translations from different perspectives such as uneven throughput allocation, considering the penalty for uneven allocations, and allocating zero throughputs for users that are all the notions of unfair resource allocation. Initially, the fairness measure f has been defined by (Jain et al. 1998) as follows:

$$f = min(P_i/P_j) \tag{2.3}$$

Where P_i and P_j are the ratio of the throughput and round trip delay received by users i and j sequentially. Jain's fairness index is the most widely used fairness evaluation technique that has been proposed by (Jain et al. 1998). The main intuition behind Jain's index is to evaluate the ratio of allocated resources to a user over other individuals in the system. Accordingly, the outcome of the evaluation is an index, preferably a value in a range between 0 to 1. Assuming a system with n users, each receives an allocation Π. In this case, the index could be formulated as follows (Jain et al. 2005):

$$J(\mathbf{\Pi}) = \frac{\left(\sum_{i=1}^{n} \Pi_i\right)^2}{n \sum_{i=1}^{n} \Pi_i^2}. \tag{2.4}$$

According to 2.4, the Jain's index evaluates the fair allocation in a system, using the index $J(\Pi)$ in which $(0 \leq J(\Pi) \leq 1)$. If the index is 1, then the system is fair so that all resources are allocated equally among users, otherwise, it is not 100% fair. In this thesis, we apply Jain's index to evaluate proposed algorithms, confirming that our approaches behave fairly among all users. Although Jain's index has been proposed to evaluate the allocation in frameworks with a single resource type, it is applicable to measure the fairness in multi-resource scenarios.

2.2.6 Cloud Computing

Cloud computing is a promising paradigm that offers scalable, on-demand, and highly available resources in a virtualized form (Namasudra et al. 2017*a*). According to Figure 2.2, cloud computing incorporates service delivery models that are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Each layer of this architecture distinguishes the level of services that must be delivered. Furthermore, cloud computing offers the private, public, and hybrid settings, known as deployment models. In this section, we take into account delivery models to explore the existing issues in deployment models.

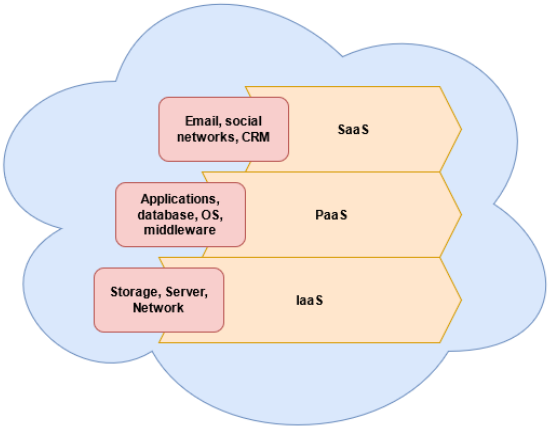


FIGURE 2.2: Cloud computing service layers

2.2.6.1 IaaS

In the IaaS delivery model (Namasudra et al. 2017*b*) that lies in the bottom layer of cloud computing infrastructure, the computational resources are provisioned on-demand concerning users' requirements in a flexible, manageable, and scalable way. The self-service feature in IaaS allows a user to access and monitor computational resources and services. The organizations may use an API or a dashboard to access cloud services. Therefore, users can get resources without requiring to set up and install any infrastructure or Software. Instead, they have control over the applications, storage, and operating system. These services are delivered as virtual resources.

2.2.6.2 PaaS

As a high-level integrated implementation, PaaS (Namasudra et al. 2017*b*) is the second layer of the cloud computing delivery model which is placed on top of the IaaS service model. It uses a web application to provide simple management of cloud resources. With IaaS, developers can deploy and customize their applications regardless of how many resources are used. Google cloud engine is an example of PaaS which provides a scalable environment to develop web applications. In PaaS, all resources are managed and operated by a third party or an enterprise. However, developers are still eligible to manage their applications. PaaS has a different delivery model compared to IaaS so that in PaaS, a platform is delivered instead of any Software. PaaS also offers a great capability to businesses to create middle-wares as highly scalable and flexible services.

There are some advantages associated with PaaS such as high availability, a reduction in the number of written codes, and a simple migration facility. As a PaaS, we use Kubernetes as a container orchestration platform to investigate the problem of fairness in resource limit assignments to pods in scheduling time.

2.2.6.3 SaaS

SaaS (Namasudra et al. 2017*b*) is the top layer in cloud computing architecture, aiming to deliver applications through the web, managed by third-party vendors. Technically, the Software is provided to end-users as a service. Software applications are provided to users through this layer via the internet/web browser. As the hardware is generally abstracted using the virtualized technology, users are not required to buy hardware to deploy their applications. The applications are accessible directly from the Internet without requiring installation. Using SaaS, organizations can easily support their services in real-time. CRM, communication tools like Skype, WhatsApp, and email are simple examples of SaaS. Furthermore, the cloud consists of private, public, and hybrid deployment models. Accordingly, We will investigate the problem of fairness in these services while the delivery models are taken into account.

2.2.6.4 Private Cloud

The computing services in the private cloud are delivered through the web and private internal network. There are many advantages of using private clouds such as scalability, self-service, elasticity, and resource customization from the specific servers (Hosny et al. 2016). In the private cloud, a set of resources are divided among users with different preferences. Typically, under the private cloud, users have access to a specific share of resources free of charge. Hence, in a mathematical perspective, users are not specified by weight and no user has an advantage over another one to receive more resources than required. Hence, without considering the delivery models, the main goal is to achieve fairness as well as maximum utilization of resources which may result in achieving the trade-off between fairness and efficiency (Khamse-Ashari 2018).

2.2.6.5 Public Cloud

Unlike the private cloud, the public cloud providers make IaaS resources available to the public for a price or free through the Internet to allow the scalable sharing of resources. The structure of the public cloud could be different according to the type of provided services. While the trade-off between fairness and efficiency matters in this case, if users pay for a service, the goal could be maximizing the revenue for providers as well as maintaining the fairness (Khamse-Ashari 2018). Taking into account the IaaS delivery model, in the public cloud, multiple types of resources are distributed in different servers and resource pools. Hence, users show considerable interest in receiving services from different servers (Farley et al. 2012). Accordingly, there are a couple of common challenging issues in the public and private clouds because of (a) heterogeneity of resources, (b) priority of tasks, dominated on a specific resource type, (c) evaluation of the fair distribution of resources, (d) establishing the trade-off between the fairness and efficiency (costs and resource utilization), and (e) satisfying desirable fairness features, specifically sharing-incentive and Pareto-efficiency. All these problems are generally common in all cloud computing delivery models. However, in the PaaS/SaaS delivery model, fairness is a significant issue in managing cloud-based platforms such as Apache Mesos (Saha et al. 2019), and Hadoop YARN fair scheduler (Lin and Lee 2016). In this thesis, we also consider the public cloud where resources are distributed across different resource pools, while users have the same weight on all resource types.

2.2.6.6 Hybrid cloud

The hybrid cloud (Vaishnnave et al. 2019) is the combination of private and public clouds. In today's computing systems, this type of cloud is being used to deliver on-demand and high-quality services to users. In particular, serving a huge number of demands only in the public cloud seems impossible. Hence, the providers tend to serve those requests using the dedicated and private servers that are normally set up on the nearest place of users. Accordingly, the authors have suggested the application of fairness to manage resources efficiently in such an environment (Madej et al. 2020). Unfortunately, almost all fair resource allocation algorithms have been implemented in simulation environments. Accordingly, we believe that it is necessary to integrate those mechanisms in hybrid-adaptable frameworks. The Apache Mesos as a well-known cluster

management platform is recently being employed in the hybrid cloud to deal with the microservice-based applications (Xue et al. 2017). The Mesos employs DRF to manage resources fairly within the cluster. Accordingly, in this thesis, we investigate the applicability of fair resources allocation algorithms in Kubernetes as a container orchestration platform that is used in the hybrid cloud. We also argue that the Kubernetes suffer from pod evictions as the Kube-scheduler does not consider resource limits in scheduling time.

2.2.7 Fairness in resource allocation and scheduling in cloud computing

Resource allocation is a fundamental problem that has been investigated in different disciplines such as operating systems, computer networks, and data center management. As can be seen in Figure 2.3, the Resource Allocation System (RAS) (Khanna and Sarishma 2015) is a mechanism to guarantee that what percentage of a resource must be provisioned to meet users' requirements. Moreover, the RAS is subjected to consider the current status of each specific resource type to select the most appropriate algorithm to allocate resources to users as well as minimizing costs.

The RAS aims to utilize and allocate a limited amount of resources due to the rapid expansion of cloud services and applications. Different factors contribute to having an optimal resource allocation in the cloud such as resource contention that could happen in a case when multiple users try to get resources simultaneously (Bilal et al. 2014). Resource fragmentation occurs when there are enough resources in the resource pool, however, resources cannot be fully utilized by applications. Therefore, it becomes more challenging in allocating a limited amount of resources among users while maintaining their satisfaction to meet QoS requirements (Abdelzahir et al. 2015). There are different types of resources such as processor, memory, bandwidth, and storage in the cloud that are available in virtual machines. This heterogeneity of resources requires developing a fair and efficient resource allocation algorithms. Therefore, any RAS system is subjected to handle some challenging issues such as the existence of multiple resource types, various users' demands, presence of multiple servers, and placement constraints (Khamse-Ashari 2018). Apart from fairness, all resources in the data center must be fully utilized to avoid resource wastage.

Besides, scheduling algorithms are employed by service providers to schedule incoming demands, aiming to manage computing resources with maximum efficiency. Task scheduling algorithms aim to maximize the profit and efficient consumption of resources in cloud computing, considering the limited amount of resources (Gawali and Shinde 2018). Hence, dozens of tasks scheduling algorithms have been proposed in cloud computing systems. The main functionality of any task scheduling algorithm is the process of receiving and mapping incoming tasks to available servers, taking into account the efficient utilization of resources. In particular, some short-term tasks in the cloud could suffer from a long waiting time in the queue that leads to the termination of those tasks. This problem becomes more challenging as the users may have requests on multiple types of resources (Singh 2014). The static task scheduling algorithms like FIFO, and Round-Robin (Boloor et al. 2010) are being used in cloud computing systems as well as heuristic optimisation algorithms such as Particle Swarm Optimization (PSO) (Zhang et al. 2015). However, these approaches do not consider multiple types of resources where the incoming tasks are dominated on a specific resource type. We will show that the recent scheduling mechanisms in terms of heterogeneous settings have fallen short in satisfying the sharing-incentive and Pareto-efficiency as desirable fairness properties.

We also show that the diversity of configurations in multiple servers may result in challenging issues in developing fair scheduling mechanisms.

2.2.7.1 Multi-resource system setting

The resource pool in the Cloud computing includes multiple heterogeneous servers, represented by $S = (1, 2, ..., s)$ S, each consists of k resource types indicated by $R = (1, 2, ..., k)$ such as CPU, memory and bandwidth. Each server is given by a capacity vector $C = (C_s^1, C_s^2, ..., C_s^k)$ where C refers to the capacity of each resource type k with respect to each server s. Assume, $U = (1, 2, ..., n)$ denotes the number of users, each has specific resource demand vector which is indicated by $X = (r_{is}^k, r_{is}^2, ..., r_{is}^k)$ of which r_{is}^k is the fraction of resource k requested by user i over any particular server. It is also assumed that all requested resources are positive ($r_{is}^k > 0$) for each user i and resource type k.

Definition 2.1. A dominant resource for each user i with respect to each server s is indicated as follows:

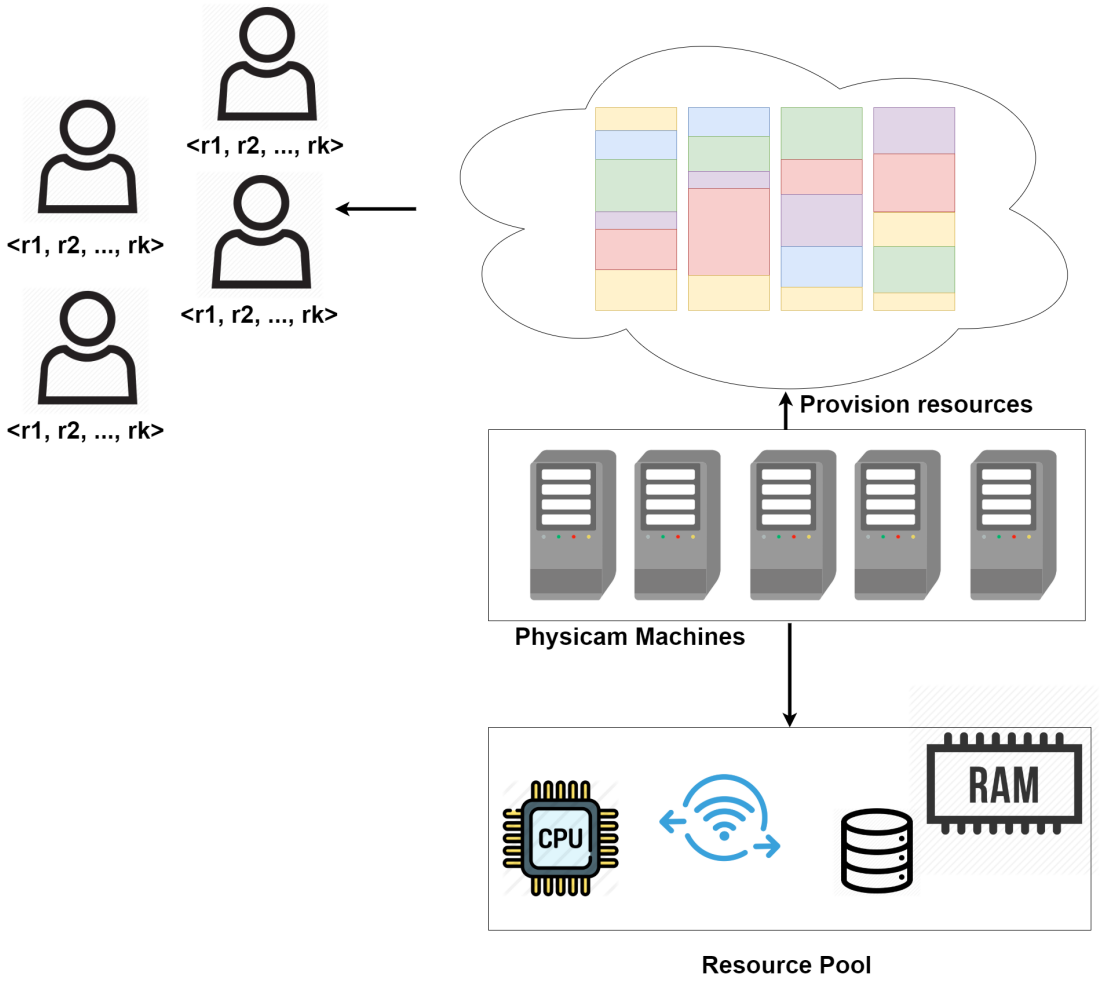


FIGURE 2.3: Resource allocation and provisioning in the cloud computing

$$d_{is}^{k} = max \frac{r_{is}^{k}}{C_{s}^{k}} \tag{2.5}$$

Given the allocation Π, the dominant share d_s is defined as the fraction of d_{is}^{k} of user i (Vakilinia 2015):

$$d_s = \frac{\Pi_i^k}{C_k} \tag{2.6}$$

2.2.7.2 Multi Resource allocation setting

In order to represent a resource allocation model in the cloud, it is assumed that there are n users and s servers, each of which has k resource type where $k \in R$. Correspondingly, the resource allocation vector could be defined as $\Pi = (\Pi_{1s}^{k}, ..., \Pi_{is}^{k})$.

Definition 2.2. A feasible allocation Π exists if no server s over-utilized with regards to its total capacity.

$$\sum_{i \in U} \Pi_{is}^{k} \leq c_{s}^{k}, \forall s \in S, k \in R \tag{2.7}$$

Definition 2.3. Given the allocation Π in server s, the maximum number of tasks Ψ that any user i can schedule, is determined as follows (Wang et al. 2013):

$$\Psi(\Pi_{is}^{k}) = \sum_{s \in S} \Psi(\Pi_{is}^{k}). \tag{2.8}$$

The allocation is preferable for all users if they can schedule more tasks in a particular server. Also, if a user utilizes a server's resources, they should not able to schedule more tasks on that server. This normally refers to a non-wasteful allocation.

Definition 2.4. An allocation Π_{is}^{k} is non-wasteful for any user i in server s, if it gets rid of any resource that decreases the number of scheduled tasks (Wang et al. 2013). In this case, the number of unscheduled tasks ${\Psi'_{is}}^{k}$ that are wasteful should be less than scheduled as ${\Psi'_{is}}^{k} < \Psi_{is}^{k}$.

2.3 Literature review

As can be seen in Figure 2.4, we categorize the notion of fairness in cloud computing in two different dimensions, including single and multiple-resource settings. In a single resource context, Max-min, proportional, and α-fairness are mainly applied in developing RAS systems. This is worth mentioning that among single resource mechanisms, the α-fairness is the most applicable approach in heterogeneous resource and server profiles. The extension of α-fairness has been applied to address the efficiency problem in DRF. The second category includes the fair mechanisms in multi-resource environments. Accordingly, four different extensions are considered, including single server, multi-server, multi-server in presence of capacity constraints, and cost-efficiency. The main focus in single server setting includes a) utility maximization for users with tasks dominated on different resource types, considering the boundaries of fair-share and resource demands while satisfying the Pareto-efficiency feature to achieve an intuitively fair allocation

(RQs 1 and 2), b) Prioritizing tasks in multiple queues based on non-identical dominant resources (RQ 3), and c) Equalizing dominant and non-dominant resources to establish trade-off between the fairness and efficiency (RQ 5). In the multi-server perspective we explore the problems regarding the sharing-incentive property, tailored to the fairness-efficiency trade-off (RQ 1) and equalization of dominant and non-dominant resources (RQ 4). In the section with placement constraints, we emphasis on equalizing the number of dominant resources in each server to reach and improve the Pareto-efficiency and sharing-incentive properties (RQs 1 and 7).

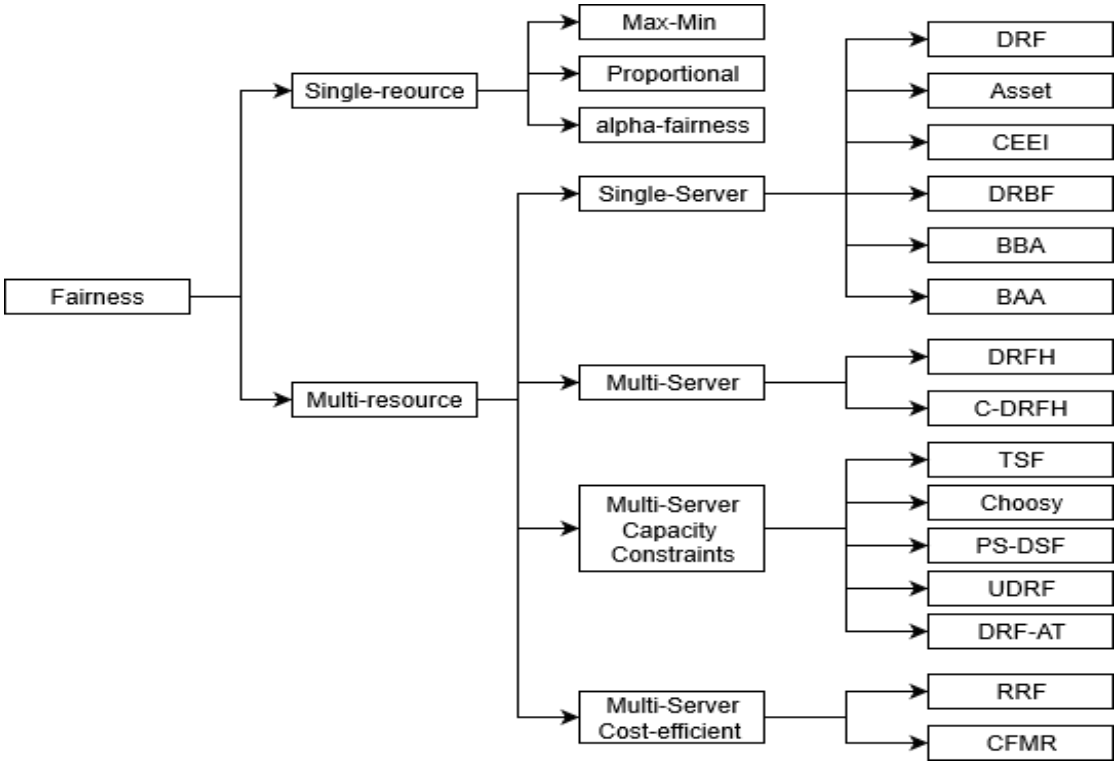


FIGURE 2.4: The evolutionary diagram of fairness in computing systems

2.3.1 Resource allocation with fairness based on single server

Resource allocation with fairness in cloud computing has been mainly considered in Apache Hadoop slot scheduler (Gautam et al. 2015). Accordingly, resource assignment to jobs in a fair way is a significant issue, especially, when a single job is executed and utilizes the whole cluster. In different time intervals, after submitting new jobs, the remaining resources in slots are allocated to new arrival jobs. Consequently, each job in the cluster benefits the same amount of CPU time. To fairly share resources among different jobs, the corresponding priorities are determined based on associated weights.

All Jobs in the scheduler are placed in a pool for scheduling by the fair share or FIFO policies. Users are not eligible to submit a large number of jobs within the cluster, as the Hadoop fair scheduler has the authority to limit the number of submitted jobs. Although Hadoop is a cloud-based framework, it considers only a single resource type. It may cause allocation problems where multiple types of resources are considered. Yet Another Resource Negotiator(YARN) (Lin and Lee 2016) is the next generation of Hadoop, aiming to address the resource sharing and scalability challenges existing in the Hadoop scheduler. Fairness and maximum resource utilization are provided by introducing the fair and capacity schedulers. Different fair scheduling policies are embedded by default in the YARN, such as FIFO, and DRF to share resources among jobs inside a queue.

Asset fairness (Ghodsi et al. 2011; Tang et al. 2018) has been proposed as a microeconomics concept to deal with the fairness problem in multi-resource environments. Under this mechanism, the valuation of equal shares is the same for all types of resources. For example, if the value of the CPU is £1, then the value of memory is £1 as well. Therefore, to allocate resources, Asset fairness aims to equalize aggregate resource values allocated to each user. Basically, for every specific user i, the Aggregate Share (AS) is indicated by $x_i = \Sigma_k R_i^k$, where K is the amount of resource r which is allocated to user i. However, asset fairness does not fulfill the *sharing-incentive* property as some users are unable to receive at least $1/n$ of resources (Wang et al. 2013). We do not target asset fairness as it does not meet the sharing-incentive as one of the most important fairness features.

A fair division of resources based on the microeconomics theory is recalled Competitive Equilibrium from Equal Incomes(CEEI) (Ghodsi et al. 2011). Under CEEI, at least $1/n$ of resources are allocated to every user. Then, a user may trade his/her resources with others in the market that is highly competitive. CEEI is Pareto-efficient and envy-free. The main idea behind CEEI is tailored to the Nash bargaining model. Based on this, only an allocation is selected that maximizes the utility of a user i given his/her allocation Π. This utility is denoted by $U_i(\Pi_i)$. Although, CEEI meets both Pareto-efficiency, and envy-free properties, it is not strategy-proof. Consequently, users could maximize their allocations by misreporting demands. Indeed, the problem of strategy-proof is still challenging in a real data-center environment. In other words, it is possible to prove this property from a theoretical perspective, nonetheless, for a real environment, it is

challenging to figure out which user tends to misreport real demands or manipulate them to get more resources.

DRF has been proposed to deal with problems in CEEI, Asset fairness, and Hadoop fair scheduler. The intuition behind DRF is to equalize dominant resources for all users. As an example, there is a system with two resource types (CPU, RAM) where two users A and B submit their tasks with demand vectors $(1CPU, 4RAM)$ and $(3CPU, 1RAM)$ respectively. If the total capacity of the resource pool is $(9CPU, 18RAM)$, then, a dominant resource for user A is RAM because of $(1/9 < 4/18)$ and for user B is CPU due to $(3/9 > 1/18)$. Accordingly, DRF allocates $(3CPU, 12RAM)$ for user A and $(6CPU, 2RAM)$ for user B. DRF has satisfied some desirable fairness properties such as the sharing-incentive, envy-free, Pareto-efficiency, and strategy proof (Ardagna and Squillante 2015). However, we will show that in some circumstances, DRF is unable to meet these properties e.g, in presence of multiple servers or long-term conditions.

As the DRF is unable to fulfill hierarchical scheduling, the authors introduced a new online multi-resource scheduler(H-DRF) in (Bhattacharya et al. 2013). The purpose of H-DRF is to reduce the possibility of starvation and makes sure that every group in the hierarchy receives a fair share of resources. The basic configuration of H-DRF is a weighted tree in which every node represents a positive value. The weights in the tree show the importance and the leaves refer to the jobs for resource allocation. Under H-DRF, it is guaranteed that every node in the tree gets a specific share of resources. H-DRF has been implemented in two static and dynamic versions where the dynamic one gives a better level of QoS in terms of response time, and job completion time 413% faster than the original DRF (Zhang and Boutaba 2014). In H-DRF, jobs within the cluster are packed efficiently that gives an excellent throughput. However, H-DRF does not show any improvement in 4-slots, and it begins to improve after 5-slots. Unfortunately, this leads to unfair, and inefficient allocations. Moreover, the naive implementation of H-DRF fails to satisfy Pareto-efficiency. Additionally, H-DRF does not consider fairness when the placement constraints are taken into place. Also, the hierarchical approach, employed in H-DRF does not show how requested resources are compared to the fair-share in resource allocation decisions (RQ 2). In particular, H-DRF allocates resources based on the proportion of demands to the fair-share. We will illustrate, comparing dominant resources with the fair-share leads to a fairer allocation as those dominant resources greater than the fair-share get exactly what they ask for. On the other hand,

others lower than that are maximized to get at least $1/n$ of resources. Furthermore, H-DRF has considered the equalization of non-identical dominant resources based on their contribution over a specific resource type (RQ 4). We will show that this type of allocation may unexpectedly maximize a specific non-dominant resource type that may result in starving other tasks, dominated on a different resource type. Consequently, this allocation is not intuitively fair (Poullie et al. 2018). Additionally, under the hierarchical and progressive-filling mechanisms, employed in H-DRF and DRF, it has not been explicitly investigated how to prioritize tasks based on non-identical dominant resources, considering a specific queuing mechanism (RQ 3). We will show that this leads to an increase in the response time for some tasks that arrive in the queue just after ones with an identical dominant resource type.

The notion of fairness with regards to users' entitlements on some bottleneck resources is not supported by DRF. Accordingly, in (Dolev et al. 2012) authors proposed a Bottleneck Based Fairness(BBF) as a new approach to allocating divisible resources. BBF targets a form of allocation as no user complains to get more resources. Hence, the allocation refers to BBF if every user gets a desirable amount of resources at least on a bottleneck one. Similar to DRF, BBF has also satisfied some desirable fairness features. However, under some conditions, it has failed to meet the Pareto-efficiency that makes it intuitively unfair (RQ 1). Although, BBF considers fairness over a set of bottleneck resources, it has ignored whether a fair allocation could affect the system's utilization. Moreover, in severe conditions, BBF has failed to satisfy the sharing-incentive and envy-free properties. To overcome this issue, in (H and P 2014) authors introduced a Bottleneck Aware Allocation(BAA) mechanism that considers a bottleneck sets of multiple resources. BAA aims to maximize overall resource utilization and fairness among users. Like DRF, BAA meets some desirable fairness properties. BAA operates in two-tiers phases that are: allocation, and scheduling. In the first phase, BAA sends out resource requests to the storage, and automatically determines the weights according to workload behaviors.

The term "trade-off between the fairness and efficiency" that has been already discussed in 1.5, is the main drawback associated with DRF. Hence, a new multi-resource allocation algorithm called DRBF (Zhao, Du and Chen 2018) has been proposed to solve this problem. DRBF has considered bottleneck resources and places them in different queues. It also employs a linear programming strategy for resource allocation based on dominant resources. DRBF provides 100% resource utilization, and better fair allocation compared

to DRF. However, we will show that like other extensions of DRF, DRBF has failed to meet the intuitive fairness as some users with dominant resources are unable to maximize their allocations. In other words, this may result in violating the Pareto-efficiency for some users, which is an obstacle against the notion of intuitive fairness (RQ 1). In particular, BAA and DRBF assume bottleneck dominant resources and equalize them based on the fair share. Additionally, it has not been explored the equalization of non-dominant resources along with dominants as bottleneck resources. Moreover, these mechanisms have ignored how the fair proportion of resources are divided among a group of users with dominant and non-dominant resources. This problem could be solved by considering the proportion of the whole resource pool for these sets of resources and equalizing them not only based on the contribution of dominant resources over a particular resource type but only based on the whole capacity of the resource pool. We will show that this solution also leads to strong trade-off between fairness and efficiency (RQ 5).We will show that the allocation under this approach could solve the problems regarding utility maximization and intuitively fair allocation.

In (Parkes et al. 2015) authors have studied the results associated with DRF in three stages. Initially, they have investigated the extension of DRF in more expressive settings in a case that users are weighted on different types of resources. Also, they have represented how it is possible to deal with indivisible resources whereas other works consider only divisible ones. The authors have claimed that DRF does not show a good performance concerning social welfare. Accordingly, they have tried to deal with this issue by studying the relationship between the desirable fairness properties, and social welfare. They also have determined the utility of each user regarding their allocations while DRF and other works focus only on basic preferences. Accordingly, if the total number of resources is k, then DRF gives a minimum of $1/k$ of optimal allocation with regards to social welfare. Therefore, it affects other fairness criteria such as SP, EF, and SI. Moreover, they have investigated that if there are zero demands in the system, how DRF could deal with those at different levels. Nonetheless, this approach has assumed that the resource pool is composed of divisible resources. Hence, if the cluster consists of a considerable number of small machines, resource fragmentation could happen.

The main research problems in this section fall into the Pareto efficiency feature and inefficient utilization of resources. Accordingly, a considerable number of users are not able to receive at least a fair share of resources. Another problem in this category is the

response time, due to the lack of a mechanism to prioritize tasks dominated on multiple resource types. Finally, violating the Pareto-efficiency, explored in the above-mentioned approaches leads to starvation for some users, may result in violating the concept of intuitive fairness. In addition to this, the problem of fairness could be more complex when users are allowed to submit multiple jobs to the data center. In this particular case, it is very difficult to judge how to share resources among users as some users may have more jobs than others. In particular, and considering the general application of fair resource allocation algorithms in the cloud, it has been assumed that all users have an equal number of jobs. Although this is still a challenging issue, it is out of the scope of this thesis.

2.3.2 Resource allocation with fairness based on heterogeneous servers

DRF and its extensions that have been explained in the previous section have assumed that all resources are concentrated in a shared resource pool. However, in real scenarios, resources are distributed in different servers with specific configurations. In such a case, the definition of a dominant resource is still challenging and unknown (Wang et al. 2013). Also, applying DRF separately in each server causes an inefficient allocation as some users may have dominant resources in multiple servers.

To overcome the efficiency problem associated with DRF in heterogeneous servers, in (Joe-Wong et al. 2012*a*) authors have proposed a mathematical approach to tackle the fairness and efficiency drawbacks corresponding to multiple types of resources, and nodes(servers). In the first step, they have introduced a mechanism to measure efficiency and fairness. In this model, resource allocation has been determined using the utility maximization problem, and a sub-gradient method regarding each server's capacity constraints. The proposed algorithms have guaranteed that fairness and efficiency have been achieved by the system. Nonetheless, under this approach, it is unknown how to define dominant resources when there are different servers with diverse specifications (Wang et al. 2013).

In (Wang et al. 2013) the generalization of DRF has been proposed, aiming to address the efficiency problem associated with DRF in heterogeneous servers. Despite (Joe-Wong et al. 2012*a*), DRFH has performed the *global dominant shares* to attain Max-Min fairness, as it guarantees service isolation among users. Technically, DRFH has

attempted to schedule all users' tasks on a single server. Althouth, DRFH has satisfied several fairness features, it is unable to guarantee the sharing incentive property in some conditions. Furthermore, DRFH has failed to satisfy bottleneck resource fairness as it maximizes the minimum global dominant shares and then schedules all tasks of a user in a single server. As we already discussed in 1.6, satisfying this important property is one of the requirements to achieve an intuitively fair allocation (RQ 1). Hence, we can say that DRFH is not intuitively fair.

In (Xu and Yu 2014) a finite extensive game-theoretic approach based on the utility functions has been introduced to solve the fairness and efficiency trade-off in heterogeneous servers. The basic purpose of the work is to pack VMs according to their utility functions for achieveing an optimal resource allocation decision on demand. Furthermore, the work has attempted to reduce resource fragmentation by scheduling VMs to appropriate servers to enhance resource utilization. To achieve this, they have considered the maximization of a particular resource type with a minimum consumption concerning each server. However, this work has not explicitly investigated the trade-off between fairness and efficiency.

According to the literature, there is still a gap to find optimal solutions in heuristic approaches. To take this into account, the authors have introduced an Ant colony-based optimization approach (Liu, Zhang, Cui and Li 2017). A new parameter has been suggested to improve the performance of the algorithm in terms of convergence. This parameter aims to control the diversity of the population in the proposed model. Based on the experiments, DRF-AT performs better in achieving *global dominant shares* compared to DRFH, and best-fit algorithms as well as higher resource utilization. However, the authors have not investigated how DRF-AT satisfies desirable fairness properties under various conditions, such as Pareto-efficiency, and sharing-incentive.

The utility functions in DRF and other similar approaches are not investigated in containers as a realistic virtualization strategy in jobs isolation. Following this, (Friedman et al. 2014) introduced a new approach called CDRF to extend DRF using a bargaining model as well as weighted max-min fairness that guarantees a strategy-proof, and efficient mechanism. The DRF policy has been further investigated in (Beltre et al. 2019). In this work, the application of DRF in the Kubernetes environment has been

explored in terms of pod scheduling, considering demands, and waiting time in a multiple nodes/servers scenarios. Nonetheless, the work has not explicitly studied the impact of resource limits in scheduling decisions. We believe that this important aspect contributes to reducing the pod evictions in scheduling time (RQ 8). Moreover, the resource limits are particularly efficient when they are associated with the namespaces for each pod, considering particular resource-intensive demands.

All the above-mentioned approaches, more specifically DRFH, have fallen short in satisfying the sharing-incentive property. As we have already discussed, failing to satisfy one of the fairness features may cause an allocation to be intuitively unfair. In particular, we believe that the concept of global dominant share could be generalized to the global aggregate resources for both dominant and non-dominant resources. This is necessary to guarantee that the entire resource pool is divided fully fair among users. We will indicate that employing this approach satisfies the sharing-incentive property which is crucial to have an intuitively fair resource allocation. Moreover, one of the most significant gaps in the current literature is that there is no explicit way to judge whether the resources in the data center is distributed among groups of users with dominant and non-dominant shares.

2.3.3 Resource allocation with fairness based on heterogeneous server and placement constraints

In the previous section, we discussed different approaches, considering multi-resource allocation with fairness in multi-server cloud. However, the recent investigations reveal that approximately 50% of submitted tasks by users are eligible to receive services from only a sub-set of servers (Sharma et al. 2011; Ghodsi et al. 2013). In other words, users' tasks could be limited to a certain number of servers. In this section, we overview several works, taking into account heterogeneous servers while the placement constraint has been considered as well.

In (Ghodsi et al. 2013) authors have proposed Constrained Max-Min Fairness(CMMF) as an extension of the Max-Min algorithm, using iterative linear programming. CMMF is an offline mechanism which is suitable for grid environments where all jobs are coarse-grained. Also, an online algorithm called choosy has been introduced to deal with a data-center environment where users enter to and leave from the system, or resources are

added or freed-up in real-time. CMMF benefits from two significant allocation features. First of all, it attempts to stimulate all resources to be pooled in a common cluster. Secondly, it tries to limit users from misreporting their constraints. The experiments conducted in the Mesos cluster driven by the Google and Facebook traces show that Choosy could converge faster to an optimal solution.

The complexity of jobs in a heterogeneous environment when each user has multiple jobs, leading to highly inefficient allocation. To address this issue, User-Dependence Dominant Resource Fairness (UDRF), has been introduced in (Tahir et al. 2015) as an extension of DRF. Moreover, an offline scheduling algorithm has been introduced that provides fairness in allocating resources among users considering Max-Min mechanism, dominant shares, and graph theory. In U-DRF, authors has investigated whether the DRF is achievable when users have multiple complex jobs. Based on this work, tasks are dependent on each other with placement constraints. Similar to other approaches, U-DRF meets some desirable fairness properties. Based on the experiments, U-DRF shows better job completion time compared to DRF, as well as reducing resource wastage.

As the strategy-proof is generally studied as one of the requirements in U-DRF, however, this important feature has been extensively investigated in presence of placement constraints by introducing Task Fair Share(TSF) (Wang et al. 2016). TSF equalizes and schedules tasks shares, considering servers' capacity constraints. Also, TSF supports dynamic scheduling to make sure that users report their demands truthfully. Similar to other approaches, TSF performs Max-Min fairness to each task share. Taking into account divisible tasks, TSF is attainable to use the progressive filling algorithm. Moreover, TSF employs an offline approach to determine resource allocation and scheduling in different stages. First of all, TSF takes task shares of users until they reach a maximum point. TSF inactivates tasks if they cannot be increased. Finally, active tasks are increased without decreasing the allocation of inactive tasks. The experiments show that TSF achieves 22% faster job completion than static approaches. Additionally, TSF isolates mini-jobs from gigantic jobs. Since TSF does not allow mini-jobs to be starved by the big jobs, they suffer from a considerable high completion time. Accordingly, we believe that a queuing mechanism should be developed to avoid such a condition in a multi-resource cloud computing environment (RQ 3).

Despite fulfilling some fairness properties, TSF does not satisfy bottleneck fairness. To

address this issue, authors in (Khamse-Ashari et al. 2017) proposed a new allocation mechanism called "Per-Server Dominant Share Fairness" as an extension of DRF that takes into account the weight of allocated resources to every individual concerning each server where the placement constraint is considered as well. PS-DSF determines virtual dominant share for each specific server. Under PS-DSF, the allocation vector is updated in certain periods to make sure that the nominated user is assigned to the most efficient server. Consequently, it achieves a higher utilization of resources compared to other allocation mechanisms such as TSF, and CDRFH. Although PS-DSF has satisfied the fairness properties like sharing-incentive and Strategy-proof, it has not met the Pareto-efficiency, which makes this allocation intuitively unfair (RQ 1). We believe that the placement constraints could be represented using a different approach, considering the equal number of non-identical dominant resources in each server as well as the maximum availability of resources (RQ 7). This strategy may result in satisfying not only the Pareto-efficiency but also the sharing-incentive property (RQ 1). Hence, incoming tasks are eligible to receive shares based on these criteria.

One of the important research directions that could be explored in the last two categories is the investigation of utility functions in real data centers in presence of heterogeneous servers. This may also help to identify utility function classes in those environments that could be further achieved by exploring the dependency among multiple resource types. Moreover, in practical scenarios, regardless of the fairness, the efficiency in data centers could be optimized. In this thesis, we only consider the utility functions in the simulation environment as well as the dependency among dominant and non-dominant resources to make sure that the resource distribution is fairly maintained. In particular, we take into account the correlation between these types of resources in terms of their contribution to the whole data center.

2.3.4 Resource allocation with fairness with cost efficient approaches

In the Cloud computing delivery model, IaaS allows users to get resources on a pay-as-you-contribute basis. Especially, Amazon EC2 enables users to get a fixed amount of computational resources on demand (Bhise and Mali 2013). This approach which is known as the T-shirt model has many negative aspects such as SLA violations, and

Algorithms	approach	SI	PE	SP	EF	comments
Asset Fairness (Tang et al. 2018)	aggregate resources		X	X	X	This algorithm does not satisfy the sharing-incentive property
CEEI (Ghodsi et al. 2011)	Competitive equilibrium based on equal incomes	X	X		X	This algorithm does not satisfy the strategy-proof property
DRF (Ghodsi et al. 2011)	Max-Min fairness generalisation and progressive filling	X	X*	X	X	Inefficient and unfair in multi server settings when is applied separately in each server * Pareto-efficiency is not satisfied in some scenarios with more than two users
DRBF (Zhao, Du and Chen 2018)	Fairness based on bottleneck resources	X	X*	X	X	* Pareto-efficiency has not been satisfied for some users
BBF (Dolev et al. 2012)	Generalisation of DRF based on bottleneck resources	X	X	X	X	Sharing-incentive and Envy-free have not been satisfied under forced conditions
DRFH (Wang et al. 2013)	The extension of DRF in multi server profiles, introducing global dominant share		X	X	X	As it maps each user's task to an exclusive server, the sharing-incentive has been violated
DRF-AT (Liu, Zhang, Cui and Li 2017)	Solving the problem of optimal heuristic solution in scheduling resources					Authors have not investigated fairness features
U-DRF (Tahir et al. 2015)	The generalisation of Max-Min fairness, graph theory, and dominant resource shares	X	X	X	X	Although it has achieved all desirable fairness properties , it is not fair as it has not consider the placement constraints
TSF (Wang et al. 2016)	Defining the fairness based on DRF and introducing task share fairness	X	X	X	X	Big jobs have been starved compared to short jobs.
PS-DSF (Khamse-Ashari et al. 2017)	The extension of DRF and TSF by introducing the virtual dominant share	X		X	X	

TABLE 2.1: The summary of multi-resource allocation mechanisms, emphasis on fairness properties such as Sharing Insentive (SI), Pareto Efficiency (PE), Strategy Proof (SP), and Envy Free (EF) as well as their main approaches.

performance degradation because of the delay in VM allocation, and run-time overhead (Nguyen et al. 2013). These negative aspects may result in inefficiency in resource utilization, and increasing operational costs. Resource allocation with fairness in heterogeneous servers is considerably interesting as the objective is to allocate resources from the low-cost servers (Khamse-Ashari et al. 2018). Therefore, in this section, we analyze the recent approaches in terms of fair and cost-efficient resource allocation in the cloud.

Reciprocal Resource Fairness(RRF) (Liu and He 2014) as a fine-grained resource allocation mechanism focused on allocating resources with fairness in a pay-as-you-go, and multi-tenant cloud. The most serious problem that exists in such computing models is *free riding* as some users could buy less amount of resources than their initial demands. Therefore, RRF tries to solve such significant issues by letting users protect their assets. Under RRF, users may trade their unused resources with other tenants. RRF targets fair allocation by proposing two approaches called *inter-tenant resource trading*, and *intra-tenant weight adjustment*. RRF also guarantees that non-dominant shares benefit a minimum amount of resources. The results show that RRF presents better performance improvement approximately by 45% on average compared to other approaches

such as the T-shirt model (Nguyen et al. 2013). VM density is also improved under the RFF mechanism among users. Moreover, resource costs in RRF are reduced by approximately 55% while the application performance degradation is lower than 15%. This represents a good trade-off between the cost and performance metrics.

A Cost-efficient and Fair Multi-Resource(CFMR) allocation algorithm in (Khamse-Ashari et al. 2018)targets minimizing operational costs in the cloud. To do this, the proposed distributed algorithm attempts to assign low-cost servers to users with fairness. The work is the generalization of (Joe-Wong et al. 2012*b*; Li et al. 2008). To maximize utility functions, the authors tried to formulate a concave optimization problem. Based on the results, CFMR reduces the operational cost by tuning α parameter.

The problem of fairness associated with the cost is out of the scope of this thesis. Hence, we consider it as one of the future research directions.

2.4 Summary

In this chapter, we study the principles of fair resource allocation algorithms in computer science and cloud computing systems. Then, we comprehensively explore the evolution of fair allocation algorithms in heterogeneous cloud environments. The fair allocation approaches are categorized in four sections as (a) single sever, (b) multiple servers, (c) multiple servers with capacity constraints, and (d) cost-efficient. Moreover, we highlight the existing problems in each category based on the research questions in 1.8. Furthermore, all existing research work is summarised in table 2.1, including fairness properties concerning each approach.

Chapter 3

Methodology

In this thesis, we use the constructive research methodology (Lukka 2003) that provides a roadmap to solve problems. It also indicates whether the research could have a contribution to the theory and real-world problems. A constructive methodology consists of constructions such as diagrams, mathematics, plans, and models. Accordingly, this thesis also provides evidence of using diagrams, algorithms, mathematical representations, and simulations as constructions. In particular, the mathematical formulas are part of the theoretical aspects of these constructions. The key stages of a constructive methodology that is applied in this thesis are illustrated in fig 3.1.

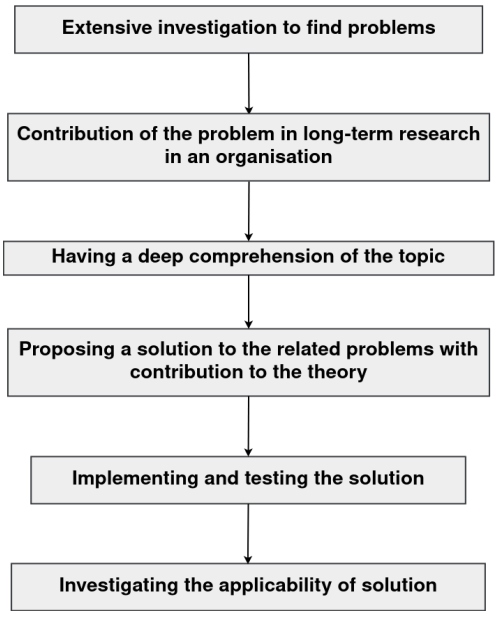


FIGURE 3.1: Key phases of constructive methodology used in this thesis

As can be seen in the figure, the first phase of a constructive methodology is finding the relevant issues as an indispensable part of any research. In this phase, the theoretical and practical concerns related to the literature should be taken into account. For this specific part, in this thesis, we conduct an extensive study on the fairness problem in cloud computing systems to find related issues, that are mostly reflected in section 2.3.

The second step of the methodology emphasizes how the research problems in the first step could have the potentials to be considered in long-term research within an organization. In this thesis, we discover significant issues based on the literature that can be extended in future cloud-based technologies. More specifically, the work in the extension of fairness in the Kubernetes framework is a long-term research goal that is intended to be applied in British Telecom's cloud computing infrastructure (see sections 4.5 and 6.3) (RQ 8).

Indeed, having a deep understanding of the topic and related theories behind the literature helps to propose appropriate solutions to overcome existing research challenges. This step is a vital point for a researcher to develop the current solutions. Accordingly, in this thesis, we consider all aspects of the previous studies in the area of cloud computing and propose suitable solutions in the form of different algorithms (see chapters 4 and 5) to solve the related problems, discovered based on the first step of the methodology. To address the existing problems associated with the recent approaches in cloud computing, this thesis proposes three novel fair resource allocation algorithms and a task scheduling mechanism. In particular, this thesis seeks to achieve an intuitively fair resource allocation (RQ 1). First, we introduce a Multi-level fair resource allocation algorithm called MLF-DRS (4.2.1). This algorithm is the combination of Max-Min and proportional fairness as two conventional fair allocation policies. Under MLF-DRS, all resource demands are judged based on the dominant shares and their boundaries associated with the fair-share function. The basic application of the fair-share has been explored in Ghodsi et al. (2011) as the main functionality of the progressive-filling algorithm. It has been also investigated in (Zhao, Du and Chen 2018) as the preliminary condition for equalizing dominant resources. As these approaches have failed to achieve the Pareto-efficiency for a certain number of users, MLF-DRS tries to solve this issue, considering the comparison of demands with the fair-share (RQ 2). Furthermore, MLF-DRS is associated with a novel scheduling mechanism that isolates incoming tasks based on dominant resources (4.2.2) (RQ 3). Next, we propose a fully fair resource allocation policy called

FFMRA (4.3.2) that considers the equalization of dominant and non-dominant resources (RQ 4). The functionality of this algorithm is different than other extensions of DRF. Technically, it employs a two-level strategy to distribute and allocate resources. Accordingly, we formulate two linear optimization problems for groups of users with aggregate dominant and non-dominant shares (see 4.3.2.1) to establish a trade-off between fairness and efficiency (RQ 5). Moreover, as Jain's index (Jain et al. 2005) is still being used to judge the fair distribution of resources, we formulate a new fairness measure to apply in multi-resource environments (see 4.3.2.3) (RQ 6). We further investigate the extension of FFMRA from a multi-server perspective, taking into account identical and non-identical server profiles. H-FFMRA (4.4.4) inherits all allocation principles, already existed in FFMRA. Similar to FFMRA, it achieves a fully fair resource allocation. In particular, the existing approaches such as DRFH and PS-DSF (Wang et al. 2013; Khamse-Ashari et al. 2017) have determined dominant resources based on the global dominant share and virtual dominant resource. Unfortunately, the sharing-incentive and Pareto-efficiency features have been violated under these mechanisms (RQ 1). This problem is addressed in H-FFMRA by determining global aggregate dominant and non-dominant resources in 4.4.4.1. Next, we propose a novel fair task scheduling mechanism called MRFS (5.2.2) due to the significant shortcomings associated with exciting task scheduling algorithms such as DRFH and PS-DSF. As it has been discussed earlier, these approaches suffer from satisfying sharing-incentive and Pareto-efficiency under specific conditions. Therefore, it motivates us to tackle these issues using a new task scheduling mechanism. Correspondingly, in MRFS we formulate a maximization problem with several constraints, employing Lagrangian multipliers. This optimization methodology tries to find an optimal solution based on the given maximization problem. We put rigorous constraints, aiming to keep the number of non-identical dominant resources in an equal state (RQ 7). To achieve this, a higher value of utility for each user is taken into account.

Before moving to the implementation phase, it is also crucial to validate how a presented solution works based on theoretical analysis. Hence, in this thesis, all algorithms and solutions are theoretically evaluated in terms of fairness properties (4.2.3.3, 4.3.3, and 4.4.6). The fifth step of the constructive methodology emphasis implementing and testing the proposed solutions. This part is considered to be one of the key aspects of the

methodology which gives proof of whether a solution is valid. Accordingly, we evaluate all proposed algorithms in the CloudSim environment (see 6.2), taking into account the allocation, resource utilization, fairness, response time, Pareto index, and sharing incentive as the basic metrics (7). All algorithms are compared to DRF and multi-host DRF. Furthermore, the evaluations are conducted, using randomly generated workloads as well as google workload traces (see 7.1.1). This phase further indicates how the aforementioned constructions have been properly illustrated to solve the existing issue(s). Correspondingly, this thesis uses almost all constructions such as algorithms, mathematical representations, diagrams, and proofs to define and represent problems.

The last stage of the methodology emphasizes investigating the applicability of proposed solutions (6.3). In particular, this allows a researcher to refer back to the previous studies and all learning processes. Therefore, this thesis explores the applicability of proposed algorithms in the context of British Telecom infrastructure (see chapter 6). Based on this, we study the fairness issue in Kubernetes as a promising container orchestration framework. Since the Kubernetes scheduler does not consider resource limits in pod scheduling time, a significant number of evictions may occur. Typically, this issue results in performance degradation in terms of response time. To deal with this significant drawback, we introduce a model to integrate our proposed algorithms in the Kubernetes scheduler (4.5.2). We also use the concept of namespaces to ensure that an accurate proportion of a node capacity is assigned as a resource limit to each pod. Then Accordingly, we formulate the maximization problem, based on proposed algorithms.

Chapter 4

Proposed fair resource allocation algorithms

4.1 Introduction

In the previous section, we comprehensively investigated the recent studies related to resource allocation with fairness in cloud computing systems. The primary focus of those works was the extension of DRF in different directions. Nonetheless, a key problem with much of the literature is that they have ignored the significance of intuitive fairness (Poullie et al. 2018). To achieve this, any resource allocation algorithm must satisfy desirable fairness features for all users and represent a fully fair allocation strategy.

In terms of the resource allocation with fairness, the concept of fair-share is a significant measure to judge the satisfaction of sharing-incentive property, as well as the Pareto-efficiency (Dolev et al. 2012). In recent approaches, such as H-DRF, the allocated resources have been calculated based on the proportion of demands over the fair-share. From another perspective, DRBF (Zhao, Du and Chen 2018) has taken the advantage of equalizing dominant resources according to the fair-share. Nonetheless, Under these mechanisms, it is not obvious whether the users are satisfied with allocated resources concerning their demand profiles. Moreover, they have not employed any decision-making mechanism to determine allocations based on the boundary of demands and fair-share. Indeed, the maximization of users' utilities with demand profiles lower than the fair-share must be maximized considerably compared to those with higher demand

than the fair-share (RQ 2). We will show that the recommended strategy could also satisfy the Pareto-efficiency for some users with dominant resources (RQ 1).

DRF and its extensions have employed the progressive filling algorithm (Ghodsi et al. 2011) in allocation decisions. Under this strategy, the minimum dominant share has been taken into account to be maximized under the notion of Max-Min fairness and conventional queuing mechanisms such as FIFO, and Round-Robin. Accordingly, there is no particular queuing strategy for isolating tasks with non-identical dominant resources (RQ 3). We will show that the response time is increased for some tasks, arriving later than a remarkable sequence of those tasks with a different dominant resource type.

Besides, it is still a challenging issue to determine whether a set of resources are divided among users in heterogeneous cloud settings. The first step is to establish a trade-off between fairness and efficiency and then determine if the resources are fairly allocated, while the desirable fairness properties are satisfied (Jin and Hayashi 2018) (RQ 5). Nonetheless, our investigations reveal that the Pareto-efficiency has not been satisfied for specific users. Consequently, it could be an obstacle against attaining such a trade-off. Second, although Jain's index (Jain et al. 2005) is a regular indicator of evaluating fairness in resource allocation, it has not been explored how to measure it in a multi-resource environment (RQ 6). This could be sophisticated when users are categorized based on demands according to the dominant and non-dominant shares.

Furthermore, the notion of fairness could be generalized to the container orchestration frameworks such as Kubernetes. The resource allocation in the Kubernetes is performed by the OS kernel according to the resource limits, assigned to each pod. As the Kube-scheduler maps pods to available nodes based on resource requests, a high possibility of pod evictions occurs. Therefore, we believe that the multi-resource fair allocation mechanisms seem a logical solution to tackle this drawback associated with Kubernetes (RQ 8).

Taking into account these problems, in this chapter, we first propose a Multi-Level Fair Resource Allocation(MLF-DRS) that allocates resources based on a decision-making approach concerning the boundaries of users' demands and the fair-share function (RQ 2). Moreover, the MLF-DRS employs a novel queuing mechanism to prioritize tasks, considering dominant resources of multiple resource types (RQ 3). To achieve a fully fair resource allocation a Fully Fair Multi-Resource Allocation called (FFMRA) is proposed.

A maximization problem is formulated, taking into account the equalization of dominant and non-dominant resources (RQ 4). Moreover, a new formulation is introduced to maintain a correlation between these types of resources to achieve a trade-off between fairness and efficiency (RQ 5). We also introduce a new fairness measure that determines the resources in a heterogeneous environment that are distributed evenly among users (RQ 6).

We further develop a model to integrate all algorithms in the Kubernetes environment by reformulating the optimization problem in terms of allocatable resources (RQ 8). Then, we extend the FFMRA to multiple server settings by introducing global aggregate dominant and non-dominant resources (RQ 5). The fairness properties evaluations indicate that our proposed algorithms satisfy almost all desirable fairness features. It is worth mentioning that our proposed resource allocation mechanisms try to solve the existing problems, in particular, Pareto-efficiency and sharing incentive features to achieve an intuitively fair resource allocation (RQ 1).

This chapter is organised as follows. Section 4.2 discusses the MLF-DRS algorithm. In section 4.3 we introduce FFMRA allocation policy. H-FFMRA is proposed in section 4.4. Section 4.5 explores the fairness and the integration of proposed algorithm in Kubernetes environment.

4.2 MLF-DRS: A Multi-Level Fair Resource Allocation Algorithm in Cloud computing

In this section, a new fair resource allocation algorithm called MLF-DRS (Hamzeh, Meacham, Virginas, Khan and Phalp 2019) is proposed. MLF-DRS employs the notion of Max-Min fairness, and proportionality. The resource allocation under MLF-DRS is determined, considering the boundaries of demand profiles and the fair-share function, considering the equalization of identical dominant resource types. Accordingly, in section 4.2.1 the allocation problem is formulated. Then, in section 4.2.2 a new queuing mechanism is introduced. In section 4.2.3 we analyse the performance of MLF-DRS, using numerical evaluations. Finally, section 4.2.3.3 evaluates fairness properties associated with MLF-DRS.

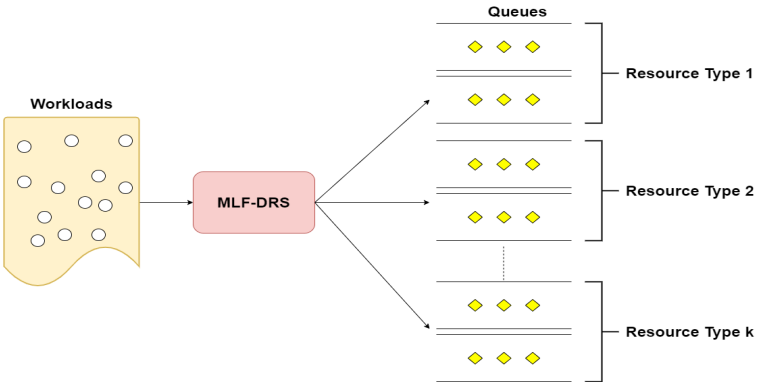


FIGURE 4.1: The architecture of MLF-DRS

4.2.1 MLF-DRS allocation

The architecture of MLF-DRS is illustrated in Figure 4.1. Once the workloads are dispatched, MLF-DRS schedules them in separate queues based on dominant, and non-dominant resources. Different sets of queues are considered concerning each resource type. In each queue, MLF-DRS applies equalization only for dominant and non-dominant resources in a specific resource type.

Given that there is a system with k types of resources presented by $R = (1, 2, \ldots, k)$ with the capacity vector $C = (c^1, ..., c^k)$, and there are n users denoted by $U = (1, 2, ..., n)$, with demand profile for each user i over resource k presented by $X = (r_i^1, ..., r_i^k)$. Without loss of generality, we consider that the requested resources are positive, as $r_i^k \geq 0, \forall i \in U, k \in R$.

Definition 4.1. Dominant and non-dominant resources that are shown by d_i^k and $\tilde{d_i^k}$ respectively, are calculated as follows:

$$d_i^k = max \frac{r_i^k}{c^k} \tag{4.1}$$

$$\tilde{d_i^k} = min \frac{r_i^k}{c^k} \tag{4.2}$$

Our assumption is that the computational resources, e.g, CPU, and RAM are divisible. Therefore, if the allocation vector for each user is specified by $\Pi = (\Pi_1^k, ..., \Pi_i^k)$, then, for the solution z and the allocation vector of each user Π_i^k, the utility function can be calculated as follows:

$$u_i(\Pi_i^k) = max(z \in^+ : \forall k \in R, \Pi_i^k \geq z.r_i^k) \tag{4.3}$$

The utility function in 4.3 is the proportion of utilized dominant resources by users. Therefore, the allocation Π is feasible if $\sum \Pi_i^k \leq c^k$.

Based on 4.1 , dominant and non-dominant profiles are specified by $D = (d_i^1, ..., d_i^k)$, and $\tilde{D} = (\tilde{d_i^1}, ..., \tilde{d_i^k})$ respectively. To calculate allocations a linear optimization problem in (3) is formulated since the optimal solution of Π is the dominant share of every user. Accordingly, the allocation Π is based on a constant proportion among a user's demand profile for multiple types of resources.

$$\begin{aligned} \text{maximize} \quad & (\Pi(d_i^k), \Pi(\tilde{d_i^k})) \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i^k \leq C^k \\ & r_i^k \leq f^k \\ & \frac{\Pi_1^k}{C^k} = \frac{\Pi_2^k}{C^k} = \cdots = \frac{\Pi_i^k}{C^k} \end{aligned} \tag{4.4}$$

Based on 4.5, each user i is able to schedule Ψ number of tasks under the allocation Π_i^k:

$$\Psi(\Pi_i^k) = \sum \Psi(\Pi_i^k) \tag{4.5}$$

Fair sharing is one of the main objectives in resource allocation and scheduling in the cloud that has received much attention in recent years. Given n number of users and k types of resources, it is assumed that each user i is entitled to a fixed ratio c^k of each resource type so that $c^1 + ... + c^r = 1$. Accordingly, each user i requests a r_i^k fraction of resource k (Dolev et al. 2012). If we assume that $r_i^k \leq c^k$, for all types of resources, if user i requests for less than his/her entitlement, consequently an appropriate amount of fair-share f^k is allocated to user i on any particular resource type. The fair-share of a resource type k without considering users' demands profiles is determined as $f^k = C^k/n$. Hence, each user could be granted at-least f^k fraction of resource k since $f^k r_i^1 + ... + f^k r_i^k \leq 1$.

Also, based on 4.6, a fair-share function f, takes into account two inputs. (a) number of users n, and (b) the capacity of each specific resource type k to decide the initial allocations for dominant resources.

$$f(n,k) = \frac{C^k}{n} \tag{4.6}$$

$$f(n,k) = \begin{cases} d_i^k = f^k & d_i^k \leq f^k, \\ d_i^k & d_i^k > f^k. \end{cases} \tag{4.7}$$

Figure 4.2, depicts the initial allocation in presence of fair-share function. Accordingly, the fair-share is initially allocated to a user with any demand as such the requested resource is less than f^k. Otherwise, r_i^k is allocated.

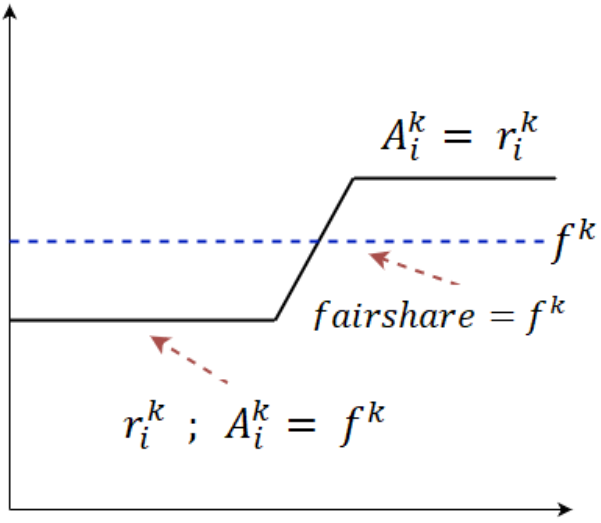


FIGURE 4.2: The fair-share function in which the allocation is determined based on the situation of the corresponding requested resource

Definition 4.2. The initial allocation A to user i is defined as follows:

$$A_i^k = \sum f^k(d_i^k) + \sum r_i^k(\tilde{d_i^k}) \tag{4.8}$$

Definition 4.3. Given that A^k is the available resource of any type of resource to user i, the allocation Π is generally determined for dominant and non-dominant resource based on the *proportionality* indicated by P as follows:

$$A_i^k = C^k - A_i^k \tag{4.9}$$

$$P = r_i^k . A_i^k / \sum_{i=1}^{n} r_i^k \tag{4.10}$$

Definition 4.4. In a perfect allocation, each user prefers to schedule more tasks. Hence, the allocation Π is non-wasteful for each user $i \in U$ as there is a positive real number y for each specific resource type k in R so that $\Pi_i^k = z.r_i^k$. Accordingly, there is another allocation $\Pi_i^{'k}$:

$$u_i(\Pi_i^{'}) > u_i(\Pi_i) \implies \forall k \in R, r_i^k > 0, \Pi_i^{'k} > \Pi_i^k \tag{4.11}$$

Definition 4.5. The allocation Π is a perfect proportional allocation if for any user i, and j, $r_i^k = r_j^j \implies i \neq j$.

The optimization problem in (4.4) is subjected to different conditions of requested resources by users which are specified in Algorithm 1. In particular, the allocation is not always equal for dominant resources as it could be varied depending on demand profiles and resource pool capacity. Therefore, the optimization problem is generally applicable when a dominant resource in a demand profile is less than or equals to f^k. Furthermore, as the type of requested resources is changed over time, there is a possibility of submitting jobs, only dominant on a specific resource type, e.g. CPU. On this occasion, the allocation is relaxed to Max-Min fairness (Ghodsi et al. 2011).

4.2.2 Queuing in MLF-DRS

In this section, we propose a new scheduler for MLF-DRS that employs multiple queues for different types of dominant resources. Despite the FIFO in which all the jobs have to wait a long time in the queue to be executed, however, in MLF-DRS as there are jobs with dominant resources on different resource types, the new scheduler considers a particular queue to nonidentical dominant resources. For example, jobs with the dominant resource in the CPU are sent to a separate queue while the jobs with dominant RAM are sent to a different one. The structure of queuing model in MLF-DRS is shown in Figure 4.1.

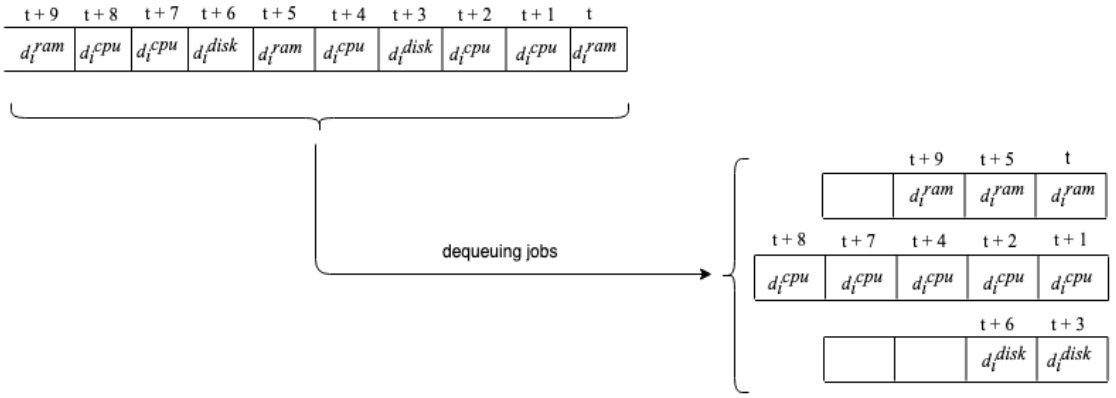


FIGURE 4.3: Dequeuing jobs from the main queue to different queues based on dominant resources

Consider time t when all tasks arrive in the main queue, indicated by vector $Q = (d^1_{iq_1}, ..., d^k_{iq_1})$, in which all tasks with different types of dominant resources are placed into it, and being processed in time interval $t = [t_0, t_{i-1}]$. If the dominant resource type of a task is denoted by d^k_i, and there are m number separate queues shown in a vector $Q = (q_1, ..., q_m)$, then the vector Q represents the distribution of tasks in the corresponding queues as $\tilde{Q} = (d^k_{iq_1}, ..., d^k_{iq_m})$.

The queuing model is also illustrated in Algorithm 1. According to the example in Figure 4.3, jobs in the main queue with dominant resources are placed in separate queues. The intuition behind the model is that each job with the same dominant resource type is placed based on the arrival time in its corresponding queue. Accordingly, in the execution time, each job in the front of each queue is executed first based on its arrival time. Hence, as can be seen in Figure 4.4, $d_i^{ram}(t)$, $d_i^{cpu}(t+1)$, and $d_i^{disk}(t+3)$ are executed first. This is a note that a job with a dominant resource on disk is scheduled (t) time earlier than the FIFO algorithm. It is more tangible if there is a job with a dominant disk and arrival time$(t+6)$. Hence, based on FIFO, it would be executed after six jobs. However, in our new queuing model, the same job is scheduled after two jobs if we assume that the first job with the dominant resource arrives in time $(t+6)$. The scheduling order considering the new queuing model is illustrated in Table 4.1.

In the worst-case scenarios where the number of dominant resources of a specific resource

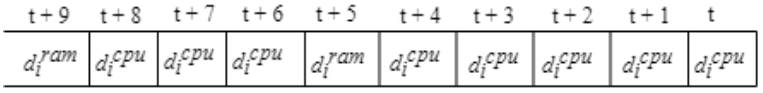


FIGURE 4.4: The order of executing tasks according to MLF-DRS scheduler

type is greater than other types of dominant resources, the chance of starvation is relatively high. It makes the scenario even worse when there are lots of dominant resources of a specific resource type in the front of a single queue. Therefore, de-queuing those jobs with a small number of dominant resources and put them in a separate queue would improve QoS and response time for those kinds of jobs.

TABLE 4.1: Scheduling order based on the new queuing model.

Time	t	t+1	t+3	t+2	t+5	t+6	t+4	t+9	t+7	t+8
Jobs	d_i^{ram}	d_i^{cpu}	d_i^{disk}	d_i^{cpu}	d_i^{ram}	d_i^{disk}	d_i^{cpu}	d_i^{ram}	d_i^{cpu}	d_i^{cpu}

Based on the example in Figure 4.4, there are two types of jobs with dominant resources on CPU and RAM. In the queue, five CPU-intensive jobs are sequentially placed in front of the queue in time interval$(t, t+4)$. However, the first RAM-intensive task arrives in time $(t+5)$. Hence, the task has to wait to be executed after the job with arrival time $(t+4)$. Accordingly, in the new queuing model, it gets executed after the job with arrival time (t) as it would be the first job in the new queue.

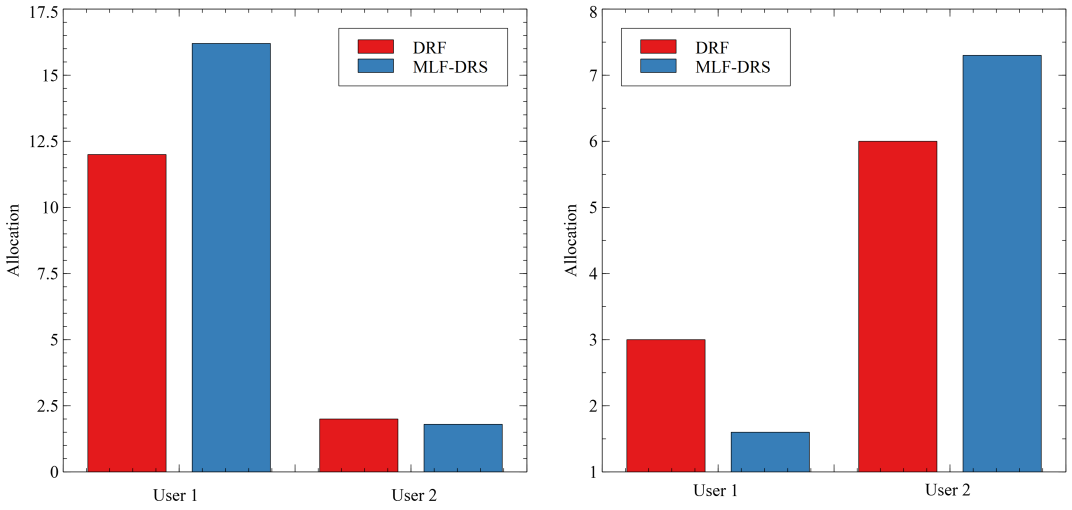
Algorithm 1 Queuing in MLF-DRS

Input: $X = (r_i^1, ..., r_i^k)$
Output: $\tilde{Q} = (d_{iq_1}^k, ..., d_{iq_m}^k)$
1: $D = (d_i^1, ..., d_i^k) \in X$
2: $t = (t_0, ..., t_{i-1})$ Time interval for incoming tasks
3: $\tilde{Q} = (q_1^k, ..., q_m^k)$ separate queues before dequeuing dominant resources from Q
4: $Q = (d^1, ..., d^k)$ The main queue, including all incoming tasks
5: **for** each d_i^k in Q **do**
6: $\quad \tilde{Q} \leftarrow d_i^k$
7: $\quad select(min(d_i^k(t)))$
8: **end for**

4.2.3 Numerical Evaluations

4.2.3.1 Numerical evaluations with two users

In this section, the performance of MLF-DRS is tested through a number of numerical evaluations. A system with two users, and resource types is considered with the capacity



(a) Memory allocation

(b) CPU allocation

FIGURE 4.5: Comparing DRF with MLF-DRS

TABLE 4.2: Comparing allocation for DRF and MLF-DRS

	User 1	User 2
Resources	CPU,RAM	CPU,RAM
Demands	1 , 4	3 , 1
DRF	3 , 12	6 , 2
MLF-DRS	1.6 , 16.2	7.3 , 1.8

vector $< 9CPUand18RAM >$. According to the example in Table 4.2 , CPU is dominant resource for users 2 and GB is dominant for users 1. Since, $f^k = 4.5$ for CPU, and the requested CPU for user 2 is less than f^k, then, the initial allocation for user 2 is $\Pi(d_2) = f^k = 4.5$. As far as the requested resource for user 1 is also less than f^k, the initial allocation is $\Pi(d_1) = f^k = 9$. Consequently, the initial allocation for non-dominant in CPU is $\Pi(\tilde{d_1}) = 1$ for user 1, and $\Pi(\tilde{d_2}) = 1$ for user 2. Hence, the final CPU and RAM allocations for both users are: $U(cpu, 1) = 1.6$, $U(GB, 1) = 16.2$, $U(cpu, 2) = 7.3$, $U(GB, 2) = 1.8$ respectively, which are shown as follows:

- **User 1:** $(CPU = 1 + ((1 * 3.5))/5.5 = 1.6$ and $RAM = 9 + (9 * 8)/10 = 16.2)$
- **User 2:** $(CPU = 4.5 + ((4 * 5.3.5))/5.5 = 7.3$ and $RAM = 1 + (1 * 8)/10 = 1.8)$

Algorithm 2 MLF-DRS algorithm

Input: $X = (r_i^1, ..., r_i^k), C = (c_1, ..., c_k)$
Output: $\Pi(d_i^k), \tilde{\Pi}(d_i^k)$

$R \leftarrow (1, 2, ..., k)$
$D = (d_i^1, ..., d_i^k) \in X$
$\tilde{D} = (\tilde{d_i^1}, ..., \tilde{d_i^k}) \in X$
$f^k \leftarrow C^k/n$
for each $d_{i,k}$ and $\tilde{d_i^k}$ **do**
    $A(d_i^k) \leftarrow f_k$ initial allocation for dominant shares
    $A(\tilde{d_i^k}) \leftarrow C_k/n$ initial allocation for non-dominant shares
end for
$S \leftarrow \sum \Pi(d_i^k)$ sum of all dominant shares of the resource type k
$S^{'} \leftarrow \sum \Pi(\tilde{d_i^k})$ sum of all non-dominant shares of the resource type k
$U \leftarrow S + S^{'}$ current utilization of resources
$Av_r^k \leftarrow C^k - U$ Available resources in the resource pool for each specific resource type
for all X **do**
    if $r_i^k(d) \leq f^k$ **and** $r_i^k(\tilde{d}) \leq f^k$ **then**
        $\Pi(d_i) = f^k + ((f^k * A_v)/U)$
        $\Pi(\tilde{d_i}) = r_i^k + ((r_i^k * A_v)/U)$
    else if $r_i^k(d) \leq f^k$ **and** $r_i^k(\tilde{d}) > f^k$ **then**
        $\Pi(d_i) = f^k + ((f^k * A_v)/U)$
        $\Pi(\tilde{d_i}) = f^k + (f^k * A_v)/U)$
    else if $r_i^k(d) > f^k$ **and** $r_i^k(\tilde{d}) \leq f^k$ **then**
        $\Pi(d_i) = r_i^k + ((r_i^k * A_v)/U)$
        $\Pi(\tilde{d_i}) = r_i^k + ((r_i^k * A_v)/U)$
    end if
end for

4.2.3.2 Numerical evaluations with four users

In order to confirm the starvation problem in DRF, a system with four users with the capacity vector $< 18CPU, 36GB >$ is taken into account. We also consider the recent proposed algorithm called DRBF.

TABLE 4.3: Resource allocation in DRF and MLF-DRS approaches

Users	User A	User B	User C	User D
Demands	3 , 1	5 , 3	1 , 5	2 , 7
DRF	6 , 2	5 , 3	3 , 12	4 , 14
DRBF	4.8 , 1.6	5 , 3	3.9 , 15.6	4.6 , 16.1
MLF-DRS	6.5 , 2.3	7.2 , 4.3	1.4 , 14.7	2.9 , 14.7

Table 4.3, indicates resource allocation in three different approaches. In this example, users A and B have a dominant resource on the CPU, while users C, and D have a dominant resource on RAM. According to the results, under DRF and DRBF policies,

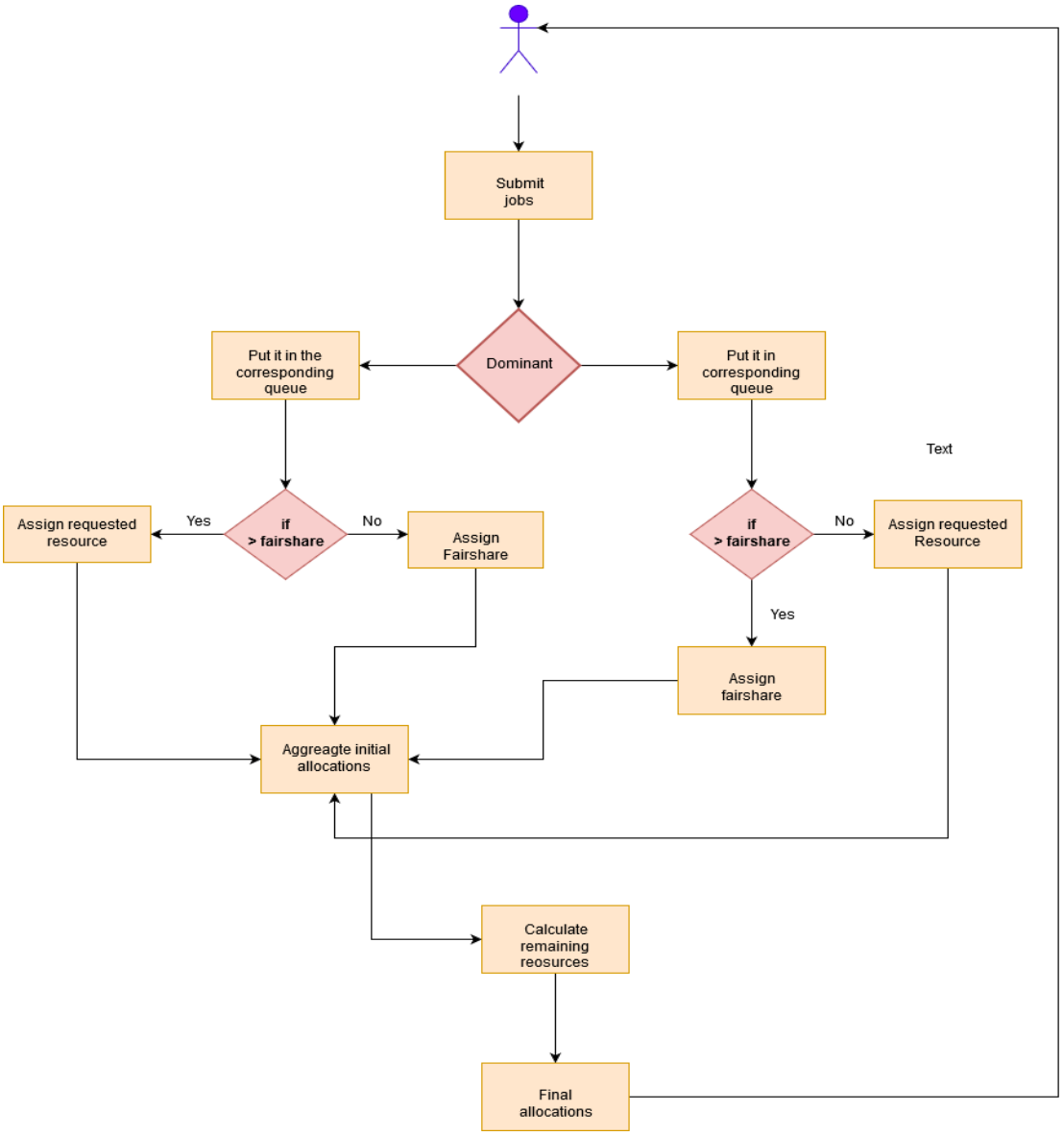


FIGURE 4.6: Resource allocation process in MLF-DRS

user B with a dominant resource on CPU is unable to maximize his/her allocation which makes DRF and DRBF intuitively unfair. Other users with non-dominant tasks on CPU, for example, users C and D, can maximize their allocations. Consequently, DRF and DRBF do not meet the Pareto-efficiency for user B. Despite these mechanisms, MLF-DRF shows an optimal allocation to users with dominant resources as it allocates less quota to non-dominants than DRF and DRBF. Moreover, the allocation is affected by the decision-making procedure in 4.6. As the requested CPU by user A is less than f^k, the allocation is slightly less than user B with the requested CPU greater than f^k. Moreover, the allocation for both users is not the same. On the other hand, users A and C get an equal share of RAM as their requested resource is less than f^k. This confirms

that MLF-DRS takes better decisions in allocating resources concerning the workload characteristics.

4.2.3.3 Evaluation based on fairness properties

In this section, the fairness properties are evaluated for MLF-DRS allocation policy.

Theorem 4.6. *MLF-DRS satisfies envy-free.*

Proof. It is assumed that k, k^* represents CPU and ram intensive tasks respectively. At the first step, each intensive task in each group k, k^* gets an equal share of available resources based on Algorithm 2 line 4 in which $f^k = C^k/n$. Hence, the allocation could be automatically envy-free. As an example, we assume that there are two intensive tasks d_i^k and d_i^{k*} with initial allocations $A(d_i^k), A(d_i^{k*})$ in the lines 6 and 7 of algorithm 2, where $A(d_i^{k*}) > A(d_i^k)$. It also takes into account that the algorithm 2 increases dominant resources proportionally based on initial allocations (see lines 20-21). Therefore, the allocation of d_i^{k*} could be greater than d_i before saturation of any resource type k. In this case, d_i^k is unable to envy d_i^*. In another case, if $\Pi(d_i^{k*}) = \Pi(d_i^k)$, MLF-DRS allocates remaining resources proportionally among users. As a result, a user with dominant task d_i^k is unable to envy a different user with a dominant task d_i^{k*}. it is worth mentioning that all combinations of this kind of allocation is that results in the envy-free property is presented in lines 23-24 and 26-27 as well. □

Theorem 4.7. *MLF-DRS meets the requirements for sharing-incentive property.*

Proof. Taking into account dominant resources for each user i, we need to show that for each allocation Π_i^k, there is $u_i(\Pi_i^k) \geq u_i(\frac{r_i^k}{c^k})$. In an equal condition when each user has a submission, dominating on a specific resource type with $(r_i^k f^k)$, then the initial allocation yields $(A_i = f^k)$ (see algorithm 2 line 6). Therefore, it is guaranteed that each user with a dominant resource get at-least $1/n$ of any distinct resource based on line 4. An allocation under MLF-DRS is called sharing-incentive when at-least one user has a demand as $(r_i^k > f^k)$. Consequently, this allocation leads to lines 26 and 27 in algorithm 2. Therefore, considering another user j with a demand $(r^{ik} \leq f^k)$ in lines 19 and 22 , there is $u_i(\Pi_i^k) > u_j(\Pi_j^k)$. Looking at the example in Table 4.2, the allocation for user B is greater than A in terms of CPU as $(\Pi_B^{cpu} > \Pi_A^{cpu})$. Accordingly, each user

gets at-least f^k which is enough to say that this allocation is satisfied sharing-incentive under MLF-DRS policy. As a result, MLF-DRS allocation policy is intuitively fair as all users with dominant resources are able to maximize their allocations without adversely affecting others.

□

Theorem 4.8. *MLF-DRS satisfies Pareto-efficiency.*

Proof. In order to show the MLF-DRS is Pareto-efficient, we need to show that $\Pi_i^k \geq \Pi_j^k$. Accordingly, user i's allocation is increased without decreasing user j's allocation. We need to prove this property in two different criteria. First we assume that $r_i^k < f^k$ and $r_j^k < f_k$. Under these conditions, both users receive exactly the same amount of resource based on the initial allocation A (see lines 6 and 7), and the final proportional allocation P (see 20, 21, 23, 24, 26, and 27 in algorithm 2). Consequently, both users' utilities are maximized. We also take into account a different condition since $r_i^k < f^k$ and $r_j^k > f_k$ like the line 25. As a result, the initial allocation A for j is r_i^j and for i is f^k. Consequently, MLF-DRS maximizes both users' allocation, taking into account the proportionality P in a same rate. This proves that both users benefit a maximum utility based on their demand profiles. Hence, we conclude that an allocation under MLF-DRS is Pareto-efficient.

□

Theorem 4.9. *MLF-DRS meets strategy-proof in which users are not able to misreport their demands.*

Proof. Assume a user with demand vectors r_i^k and $d_i'^k$ in which r_i^k and $dr_i'^k$ denote true and misreported demands respectively. Given that MLF-DRS increases dominant resources based on available resources in each stage, if a user with r_i^k tries to manipulate the server by$r_i'^k$ and considering the capacity constraint in 4.4, in that case, the constraint could be violated by misreporting the true demand by the user. Hence, a user is unable to misreport his/her demand under the MLF-DRS allocation policy. In particular, MLF-DRS considers different queues for non-identical dominant resources 4.2.2 (see also line 3 in algorithm 1). As the consequences of dequeuing approach (line 6), and in the first step, the possibility of misreporting demands is reduced to $1/\Psi(q)$ in which $\Psi(q)$ denotes

the number of queues where the tasks dominated on specific resource types are taken place. This is due to that a task with a distinct dominant resource type is unable to manipulate others' allocations with a different dominant resource type. Accordingly, each task is served based on the arrival time (line 2). Let's assume a user i with a dominant resource d, and user j with a different dominant resource d'. Under MLF-DRS, if users i and j arrive at times t and $t+1$ respectively, user i's task is finished just before user j's task is started. Consequently, user j is unable to manipulate user i's allocation by misreporting demands. Therefore, MLF-DRS satisfies the strategy-proof feature.

□

4.3 A Fully-Fair Multi Resource Allocation Algorithm in Cloud Environments (FFMRA)

In this section, a new fair allocation mechanism called FFMRA (Hamzeh, Meacham, Khan, Phalp and Stefanidis 2019) is introduced. FFMRA employs a two-hierarchy approach as in the first step, it determines the fair distribution of resources among groups of users with dominant, and non-dominant shares. It calculates the proportion of aggregate demands to the whole capacity of the resource pool. Then it applies MLF-DRS to allocate resources for each user in a corresponding group. The main purpose of FFMRA is to formulate two optimization problems to equalize both dominant and non-dominant resources. Furthermore, proposing a correlation between dominant, and non-dominant resources to achieve a strong trade-off between fairness and efficiency.

In section 4.3.1, the motivation behind FFMRA with examples is explored. In section 4.3.2 we propose and formulate FFMRA. Section 4.3.2.3 introduces a new fairness measure in multi-resource environments. Finally, in section 4.3.3, the fairness features associated with FFMRA are evaluated.

4.3.1 Motivation

We already highlighted the intuitive fairness problem associated with DRF and its recent developments. DRF considers only dominant resources and equalizes them to calculate allocations. This way of resource distribution may lead to an inefficient and imbalanced allocation. For example, let's assume some users with dominant and non-dominant resources on CPU and RAM. To achieve a balanced allocation, both groups of users with dominant and non-dominant resources should get an equal proportion of the resource pool.

To investigate the problem more extensively, in the first step, we go through an example, considering the DRF policy. According to the example in table 4.4, there is a system with two types of resources: CPU, and RAM, where two users A, and B schedules their jobs, using demand vectors (1 CPU, 4 RAM), and (3 CPU, 1 RAM) respectively. Under the DRF policy, users A and B receive $(3CPU, 12RAM)$, and $(6CPU, 2RAM)$ respectively. Correspondingly, users with dominant resources on CPU and RAM, get the

same proportion of resource pool which is 2/3, while the proportion for non-dominant resources is not the same since ($3/9 \neq 2/18$). This example confirms that DRF is unable to utilize all resources as 4GB of RAM is wasted.

In scenarios with more than two users, the even distribution of resources dominated in a specific resource type becomes more intriguing as some users are unable to maximize their allocations. Therefore, to explore the issue more comprehensively, we refer to DRBF algorithm (Zhao, Du and Chen 2018). The example in Table 4.7, illustrates the allocation in DRF and DRBF. According to the table, under DRBF policy, user B with a dominant resource on CPU with resource demand of 5, unable to get more resource in contrast with user A with the demand of 3 on CPU which is also dominant. The main reason behind these allocations is that both mechanisms attempt to maximize a user's utility with the lowest dominant resource that lies in the main assumption of Max-Min fairness algorithm (Taddei 2015). However, employing Max-Min fairness along with a progressive filling algorithm that has been used in almost all approaches, may cause starvation.

To investigate the above-mentioned issue, the total allocation for groups of users is aggregated with dominant and non-dominant shares in each specific resource type to figure out how resource pool capacity is distributed evenly among these groups. In the first step, the sum of allocations for dominant and non-dominant CPU in DRBF is calculated as $4.8 + 5 = 9.8$ and $3.9 + 4.6 = 8.5$ respectively. Similarly, for dominant and non-dominant RAM, the total shares are $15.6+16.1 = 31.7$ and $1.6+3 = 4.6$ in sequence. The allocations for non-dominants are substantially high in contrast to the allocations for dominant shares in both resource types. To determine whether the allocation is balanced or not, it is fundamental to determine the proportion of allocations for dominant shares concerning each particular resource type. As the resource pool capacity for both CPU and RAM are 18 and 36 respectively, the proportion of CPU for all users with dominant shares is $9.8/18 = 0.54$, while for dominant RAM shares is $31.7/36 = 0.88$ of which $0.54 \neq 0.88$. On the other hand, considering non-dominant CPU and RAM shares, the proportions are calculated as $8.5/18 = 0.47$ and $4.6/36 = 0.12$ of which $0.47 \neq 0.12$.

The calculations confirm that DRBF considers the most proportion of the resource pool to users with dominant shares on RAM. This may result in distributing the low amount of resources to users with dominant shares on CPU, leading to unfair allocation with

regards to intuitive fairness. DRF behaves the same as DRBF so that the proportions of a resource pool to users with dominant shares on CPU and RAM are determined as $11/18 = 0.61$ and $26/36 = 0.72$, respectively. Corresponding to non-dominant CPU and RAM shares, the proportions are calculated as $7/18 = 0.38$ and $5/36 = 0.13$, of which $0.38 \neq 0.13$. So, DRF unable to provide a fair distribution of resources. These analyses also indicate that taking into only dominant shares is not sufficient to claim that an allocation mechanism is fair. Moreover, these examples indicate that the allocation under DRF and DRBF is intuitively unfair. This motivates us to propose a new fair resource allocation algorithm called FFMRA in cloud computing systems. We believe that all resources in the resource pool are subjected to be distributed equally among users.

TABLE 4.4: The allocation of resources in DRF and FFMRA with resource capacity (9 CPU, 18 RAM)

Users	User A	User B
Resources	(CPU , RAM)	(CPU , RAM)
Requested	(1 , 4)	(3 , 1)
DRF	(3 , 12)	(6 , 2)
FFMRA	(2 , 14)	(7 , 4)

4.3.2 FFMRA

FFMRA is a generalization of DRF. The intuition behind FFMRA is to equalize both dominant and non-dominant resources. The architecture of FFMRA is depicted in Figure 4.7. As a scheduler, FFMRA attempts to de-queue all tasks with different demand profiles and put them in separate queues concerning their specific dominant resources. Accordingly, each task is placed in its specific queue corresponding to its dominant resource type. then, MLF-DRS is employed to finalize the allocation among the group of users in each specific resource type. To understand how FFMRA works, we refer to the example in the previous section where two users and two resource types. Initially, FFMRA calculates the contribution of dominant and non-dominant resources in the resource pool. This gives a good correlation between both types of resources which also helps to keep the system in a balanced condition to achieve a strong trade-off between fairness and efficiency.

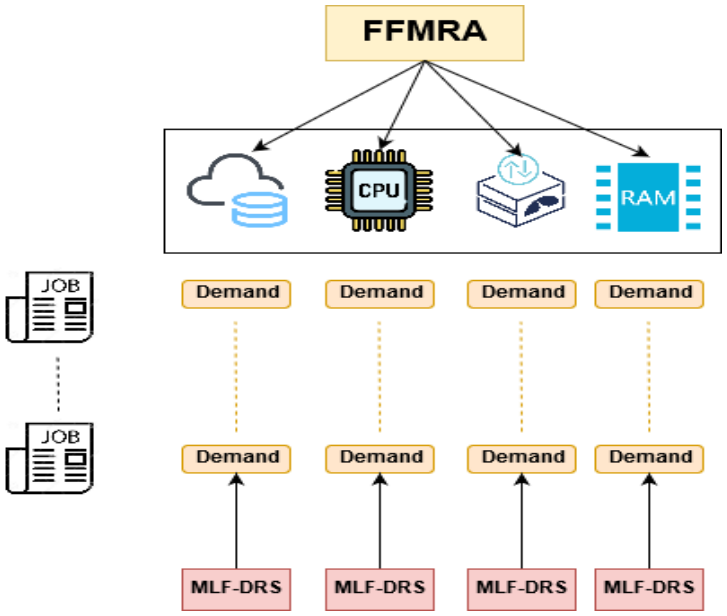


FIGURE 4.7: FFMRA architecture

4.3.2.1 Problem formulation

Definition 4.10. Consider a system with different types of resources, indicated by $M = (1, 2, ..., k)$ with the capacity vector $C = (c^1, ..., c^k)$, as well as n number of users presented by $U = (1, 2, ..., n)$ with demand vector $X = (r_i^1, ..., r_i^k)$. If Ψ denotes the number of tasks that each user can schedule, using demand vector X, then, the following constraint must be satisfied:

$$\sum_{i=1}^{n} \Psi_i^k / r_i^k \leq C^k \tag{4.12}$$

Given that $D = (d_i^1, ..., d_i^k)$ and $\tilde{D} = (\tilde{d}_i^1, ..., \tilde{d}_i^k)$ represent vectors for dominant and non-dominant resources of k resource types; d_i^k and $\tilde{d}_i^k$ are calculated as follows:

$$d_i^k = max(\frac{r_i^k}{C^k}) \tag{4.13}$$

$$\tilde{d}_i^k = min(\frac{r_i^k}{C^k}) \tag{4.14}$$

Where r_i^k demonstrates resource type k requested by user i with a demand vector which is always positive ($r_i^k > 0$). C^k also shows the maximum capacity of resource type k. If the total capacity of the resource pool(The sum of the maximum capacity of all

resources) is indicated by T_C^k, then the proportion of total resources for dominant and non-dominant resources in 4.13 and 4.14, specified by $\rho(d_i^k)$ and $\rho(\tilde{d_i^k})$ are calculated as follows:

$$\rho(d_i^k) = \frac{\sum d_i^k * T_C^k}{\sum d_i^k + \sum \tilde{d_i^k}} \tag{4.15}$$

$$\rho(\tilde{d_i^k}) = \frac{\sum \tilde{d_i^k} * T_C^k}{\sum d_i^k + \sum \tilde{d_i^k}} \tag{4.16}$$

Hence, according to 4.15 and 4.16, the allocation for each user is calculated based on the following maximization problem:

$$\begin{aligned} \text{maximize} \quad & (r_i^1, ..., ..., r_i^k) \\ \text{subject to} \quad & \sum_{i=1}^{n} r_i^k \leq C^k. \\ & \sum_{i=1}^{n} d_i^k \leq \rho(d_i^k). \\ & \sum_{i=1}^{n} \tilde{d_i^k} \leq \rho(\tilde{d_i^k}). \\ & r_i^k \geq 0, \forall n \in U \\ & r_i^k = 0, \forall n \notin U \end{aligned} \tag{4.17}$$

Formulas 4.15 and 4.16 indicate a correlation between dominant and non-dominant shares as, $\rho(d_i^k)$ and $\rho(\tilde{d_i^k})$ are determined based on the contribution of both shares in the resource pool. This correlation, taking into account the maximization problem in 4.17 establishes a trade-off between fairness and efficiency if the given constraints are satisfied. This means that, while the full utilization of resources is maintained, a fully fair allocation is also achieved under FFMRA.

Definition 4.11. An allocation Π_i^k is feasible if it meets the constraints and conditions in 4.17.

Definition 4.12. It is said that a resource k is *bottleneck* if :

$$\frac{r_i^k}{C^k} \geq \frac{r_i^{k'}}{C^{k'}}, \forall k', i \in U \tag{4.18}$$

In this case, each allocation is subjected to be reduced to *Max-Min* fairness.

Definition 4.13. FFMRA, is satisfied by allocation Π_i^k if it is possible, and Π_i^k as the shared resources to a user i must be increased with decreasing Π_i^k for another user j.

Lemma 4.14. *Consider a resource pool with resource types k where there are n users, and d_i^k denotes a dominant resource for user i. Then, $\Psi = \frac{C^k}{\Psi r_i^k}$. In particular, the total number of tasks presented by Ψ is determined by dividing the capacity of a specific resource type over the number of requested resources.*

Proof. The minimum amount of resource type k received by user i which is represented by $\rho(d_i^k)$, guarantees that a user with a dominant share d_i^k receives at least C^k/n which is the fair share f^k of any specific resource type k. Given that $\Psi = \frac{C^k}{\Psi r_i^k}$ (Wang et al. 2013) indicates the minimum number of tasks that a user can schedule in a server. Furthermore, it is assumed that the demand vector is denoted by $X = (r_i^1, ..., r_i^k)$. Accordingly, r_i^k shows a fraction of dominant resource d_i^k by user i. Therefore:

$$\Psi.r_i^k = \frac{C_k}{n} \implies \Psi = \frac{C^k}{(\Psi r_i^k)} \tag{4.19}$$

□

Based on Definition 4.13, the allocation mechanism subjects to satisfy the following fairness requirements:

- ***Pareto-efficiency***: Given that there are n users in which $i \in U$, then, the number of tasks allocated to each user i with a demand profile r_i^k, must be increased with decreasing the allocation of another user $u^{'}$, $u \neq v$ (Tahir et al. 2015).

- ***Envy-free***: A user cannot envy another user's allocations:

$$\Psi d_i^k(\Pi_i^k) \geq \Psi d_i^k(\Pi_{i'}^k), \forall i, i^{'} \in U, i \neq i^{'} \tag{4.20}$$

  Where $\Psi d_i^k(\Pi_i^k)$, and $\Psi d_i^k(\Pi_{i'}^k)$, indicate the number of tasks that users i, and $i^{'}$ can schedule respectively based on dominant resources.

- ***Sharing-incentive***: This property confirm that a user is unable to better of others' allocations when all resources are fairly divided. So that:

$$\Psi d_i^k(\Pi_i^k) \geq \Psi d_i^k(c^k/n), \forall i \in U. \tag{4.21}$$

- ***Strategy-proof***: A user is unable to increase his/her allocation by misreporting demands:

$$\Psi r_i^{'k} \ \leq \Psi r_i^k \ , \forall i \in U, \forall r^{'} \neq r. \tag{4.22}$$

Definition 4.15. Under FFMRA mechanism, the utility of a user is the allocated resource with respect to her dominant resource. Hence, given u_i^k as the utility of resource type k and d_i^k the dominant resource of user i, if $u_i^k > d_i^k$, then it means that a user gets the highest utility under allocation Π_i^k.

Definition 4.16. The utility U_i^k of user i under allocation Π_i^k is the fraction of $\rho(d_i^k)$, and $\rho(\tilde{d_i^k})$ that a user receives from the whole resource pool.

$$U_i^k = max \sum_{i=1}^{n} \Pi_i^k \tag{4.23}$$

4.3.2.2 Example

Consider the same example depicted in Table 4.4, including two users A and B with resource pool capacity $(9CPU, 18GB)$. Intuitively, FFMRA starts with summing up dominant resources for both users A and B. Hence, in the first step, the sum of dominant and non-dominant resources are 7 and 2 respectively. Then, the sum of all resources in the resource pool is $9+18 = 27$. Secondly, FFMRA calculates the proportion of resource pool for both dominant and non-dominant resources. For dominants, the value for P is determined to $(27 * 7)/9$ and for non-dominants it is $P = (27 * 2)/9 = 21$. Accordingly, the next step is to divide the value of P for each specific resource type. Therefore, for dominant and non-dominant CPUs, we have $(21 * 9)/27$ and $(6 * 9)/27$. For dominant RAM, there are $(21 * 18)/27$ for dominants and $(6 * 18)/27$ for non-dominants. Finally, MLF-DRS is applied to achieve FFMRA allocation.

4.3.2.3 Fairness measure

To evaluate the fairness, a new measure is proposed that calculates the fairness using a balanced index β among a group of users in each specific resource type. β confirms that how resource pool capacity is distributed fairly among the users regardless of dominant and non-dominant resources. A system is called fully fair if β tends to zero value. β is determined based on the differences of total allocated resources to dominant and non-dominant shares in each particular resource type. Under allocation Π_i in a system with resource capacity C_k, the fairness(balance) index is calculated for both dominant and non-dominant resources denoted by β_d and $\beta_{\tilde{d}}$ respectively as follows:

$$\beta_d = \left|\left(\sum(\frac{\Pi_i(d_i^1)}{C^1})\right) - \left(\sum(\frac{\Pi_i(d_i^2)}{C^2})\right) - ... - \left(\sum(\frac{\Pi_i(d_i^k)}{C^k})\right)\right| \in (0,1) \tag{4.24}$$

$$\beta_{\tilde{d}} = \left|\left(\sum(\frac{\Pi_i(\tilde{d_i^1})}{C^1})\right) - \left(\sum(\frac{\Pi_i(\tilde{d_i^2})}{C^2})\right) - ... - \left(\sum(\frac{\Pi_i(\tilde{d_i^k})}{C^k})\right)\right| \in (0,1) \tag{4.25}$$

Based on 4.24 and 4.24 the value of β falls into a small number between 0 to 1. Hence, based on Table 4.5, β is 0 for dominant and non-dominant resources under FFMRA which indicates that it is totally fair. However, according to Table 4.6, β is approximately 0.22 under DRF policy for non-dominant resources, implying unfair resource allocation. In real environments, β could be a reasonable metric to indicate the fairness.

TABLE 4.5: The allocation and utilization of resources in FFMRA where user 1 has dominant share in RAM and user 2 has dominant share in CPU.

	CPU	RAM
User 1	0.222222	0.777778
User 2	0.777778	0.222222
utilization	1	1

TABLE 4.6: The allocation and utilization of resources in DRF where user 1 has dominant share in RAM and user 2 has dominant share in CPU.

	CPU	RAM
User 1	0.333333	0.666667
User 2	0.666667	0.111111
utilization	1	0.777778

In order to determine the fairness among a group of users in each specific resource type, we rewrite the *Jain's fairness* index in 2.2.5 as follows:

$$f(d_i^1, ..., d_i^k) = \frac{(\sum d_i^k)^2}{n \sum (d_i^k)^2} \tag{4.26}$$

$$f(\tilde{d}_i^1, ..., \tilde{d}_i^k) = \frac{(\sum \tilde{d}_i^k)^2}{n \sum (\tilde{d}_i^k)^2} \tag{4.27}$$

The formulas in 4.26 and 4.27 calculate the fair index for dominant and non-dominant resources respectively.

4.3.2.4 A scenario with two users

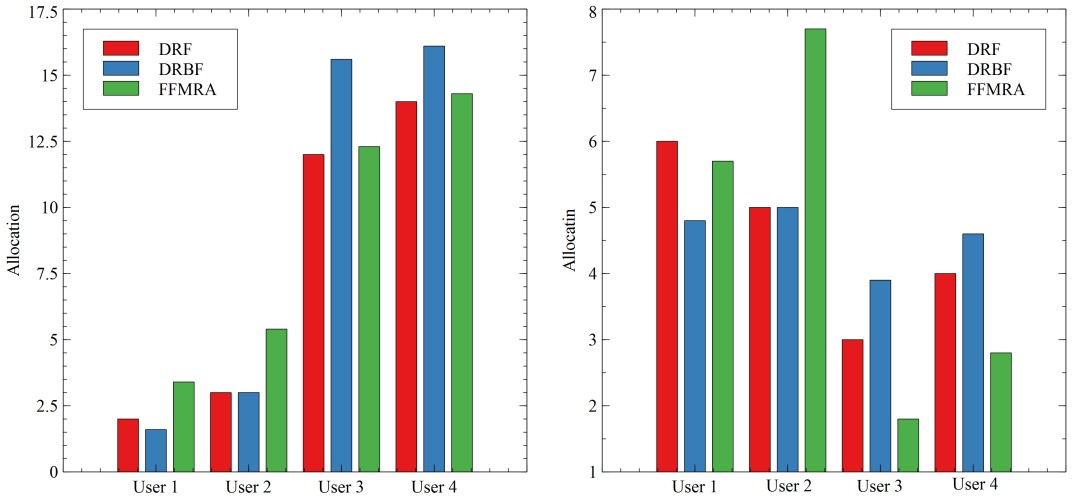
According to the numerical representation in Table 4.4, if it is considered that i_1, i_2, j_1, j_2 are the number of tasks allocated to both users and capacity of the resource pool is (9 CPU, 18 RAM), users A, and B receive $(i_1, 4i_2)$ and $(3j_1, j_2)$ respectively. Therefore, taking into account Algorithm 2, the aggregate of demands with dominant resources is 7(four dominant tasks on RAM for user A and three tasks with dominant CPU for user B) which is 2 for non-dominants(one non-dominant task in CPU for user A and one non-dominant task in RAM for user B). As far as the total number of resources in the resource pool is 27, and, concerning proportionality, dominant and non-dominant shares get 21/27 and 6/27 of the resource pool respectively.

Hence, by specifying the proportion of dominant and non-dominant shares, we are ready to calculate the allocation for each user with the following problems:

$$\begin{aligned} &\text{maximize} && (i_2, j_1) \\ &\text{subject to} && 4i_2 + 3j_1 \leq 21. \end{aligned} \tag{4.28}$$

$$\begin{aligned} &\text{maximize} && (i_1, j_2) \\ &\text{subject to} && i_1 + j_2 \leq 6. \end{aligned} \tag{4.29}$$

Solving 4.28 and 4.29 gives $i_1 = 3.5, j_1 = 2, i_2 = 2.3$ and $j_2 = 4$. Hence, the final allocations are determined as $(2CPU, 14RAM)$ for user A and $(7CPU, 4RAM)$ for user B. Accordingly, compared to DRF(see Table 4.4), not only the allocation is fair, but also the utilization is 100% for both resources. Moreover, FFMRA utilizes 100% of resources and outperforms DRF in terms of resource utilization as DRF utilizes only 14/18 of



(a) Memory allocation

(b) CPU allocation

FIGURE 4.8: Comparing FFMRA with DRF and DRBF in allocating resources

RAM and leaves 4/18 unused whereas FFMRA utilizes 18/18 of RAM. Furthermore, DRF allocates more CPU to user A with the non-dominant resource. Consequently, user B with the dominant CPU receives 6 CPU units while FFMRA allocates only 2 units of CPU to user A and 7 CPU for user B. This is important to note that, DRF is unable to offer a maximum amount of resources to user A with the dominant resource in RAM. Consequently, it allocates only 12GB of RAM to that user, whilst FFMRA allocates 14 GB RAM to user A and 4 CPUs for user B with the non-dominant resource in RAM. Tables 4.5 and 4.6 indicate how FFMRA maintains fairness and efficiency. Overall, the numerical evaluations confirm that FFMRA not only achieves intuitive fairness but also shows higher resource utilization than DRF policy.

4.3.2.5 A scenario with more than two users

As a general solution, for a system with many users, the allocation can be determined using Algorithm 3 which gives an alternative way to calculate resource allocation. This is a note that after distributing resources among all users with dominant and non-dominant shares, the allocation for each user is relaxed to proportional sharing which is specified in lines 13 and 14 of Algorithm 3.

Algorithm 3 FFMRA allocation

Input: $D, \tilde{D}, C$
Output: $\rho_c(D), \rho_c(\tilde{D}), \Pi(d_i^k), \Pi(\tilde{d_i^k})$

$T_c \leftarrow \Sigma C$ sum of the capacity of all resources
$X = (r_i^1, ..., r_i^k)$
$U \leftarrow (1, 2, ..., n)$ total users in the system
$C = (c^1, ..., c^k)$
$D = (d_i^1, ..., d_i^k) \in X$
$\tilde{D} = (\tilde{d_i^1}, ..., \tilde{d_i^k}) \in X$
$SD \leftarrow \Sigma D$ the sum of all dominant resources
$S\tilde{D} \leftarrow \Sigma \tilde{D}$ the sum of all non-dominant resources
$\rho(D) \leftarrow T_c * SD/SD + S\tilde{D}$ The proportion of total resource pool capacity for dominant shares
$\rho(\tilde{D}) \leftarrow T_c * S\tilde{D}/SD + S\tilde{D}$ The proportion of total resource pool capacity for non-dominant shares
$\rho_c(D) \leftarrow (\rho(D) * c_i)/T_c)$ The proportion of whole dominants to each identical dominant resource type
$\rho_c(\tilde{D}) \leftarrow (\rho(\tilde{D}) * c_i)/T_c)$ The proportion of whole non-dominants to each identical non-dominant resource type
$S \leftarrow \sum d_i^k$ the sum of identical dominant resources
$S' \leftarrow \sum d_i^k$ the sum of identical non-dominant resources
$Av_d^k \leftarrow \rho_c(D) - S$ current available resources to identical dominants
$Av_{\tilde{d}}^k \leftarrow \rho_c(\tilde{D}) - S'$ current available resources to identical non-dominants
$f_k = C^k/n$ fair-share
for all X **do**
  if $r_i^k(d) \leq f^k$ **and** $r_i^k(\tilde{d}) \leq f^k$ **then**
    $\Pi(d_i) = f^k + ((f^k * A_v)/U)$
    $\Pi(\tilde{d_i}) = r_i^k + ((r_i^k * A_v)/U)$
  else if $r_i^k(d) \leq f^k$ **and** $r_i^k(\tilde{d}) > f^k$ **then**
    $\Pi(d_i) = f^k + ((f^k * A_v)/U)$
    $\Pi(\tilde{d_i}) = f^k + (f^k * A_v)/U)$
  else if $r_i^k(d) > f^k$ **and** $r_i^k(\tilde{d}) \leq f^k$ **then**
    $\Pi(d_i) = r_i^k + ((r_i^k * A_v)/U)$
    $\Pi(\tilde{d_i}) = r_i^k + ((r_i^k * A_v)/U)$
  end if
end for

TABLE 4.7: Resource allocation in three different algorithms with resource capacity (18 CPU, 36 RAM)

Users	User A	User B	User C	User D
Demands	3 , 1	5 , 3	1 , 5	2 , 7
DRF	6 , 2	5 , 3	3 , 12	4 , 14
DRBF	4.8 , 1.6	5 , 3	3.9 , 15.6	4.6 , 16.1
FFMRA	5.7 , 3.4	7.7 , 5.4	1.8 , 12.3	2.8 , 14.3

Taking into account Algorithm 3 and also, Table 4.7, the proportion of allocation with respect to the capacity of each resource type for the tasks with dominant and non-dominant CPU and RAM is $13.4/18 = 0.74$ and $26.6/36 = 0.74$ respectively of which $0.74 = 0.74$. For the tasks with non-dominant CPU and RAM, we have $4.6/18 = 0.25$ and $8.8/36 = 0.25$ of which $0.25 = 0.25$. Therefore, FFMRA maintains the balance in resource allocation by distributing resources evenly among the group of users. A balanced allocation is obviously seen in Figure 3.8 for both CPU and RAM allocations. Based on the figure, DRBF and DRF consider more resources to users with dominant RAM. However, users with dominant CPUs get a relatively lower rate compared to RAM allocation. While, under FFMRA mechanisms, allocations are maximized proportionally and fairly among the users. Moreover, the numerical analysis reveals that FFMRA keeps the system in a balanced condition and allocates resources fairly compared to DRF and DRBF.

4.3.3 Fairness properties analysis

In this section, it is explored that how FFMRA could meet some desirable fairness properties.

Lemma 4.17. *For each user $i \in U$, any allocation Π_i^k is possible under FFMRA if there is at least one bottleneck resource.*

$$\exists k \in R, \sum \Pi_i^k = C^k \tag{4.30}$$

Proof. By contradicting the given assumption in 4.19, we suppose that there is a user i with no bottleneck resource on resource type k. Therefore:

$$\forall k \in r_i^k \sum \Pi_i^k < C^k \tag{4.31}$$

4.20 confirms that user i has not any submission which is bottleneck on a specific resource type k. Under this assumption, FFMRA continuously allocates resources regardless of any bottleneck resource. Hence, this contradicts the main hypothesis in 4.19. Therefore, FFMRA satisfies bottleneck fairness.

□

Theorem 4.18. *FFMRA satisfies envy-free.*

Proof. Assume that D represents a set of all dominant resources in entire system. If $R = (r_i^1, ..., r_i^k)$ indicates each specific resource type such as CPU, and RAM. Based on the Algorithm 3 FFMRA considers equal proportion of the resource pool for each specific resource, considering $\rho(d_{i,k})$ (see line 9 and 10). Hence, as an example, for each specific resource in R we assume r_i and $r_i^{'}$ are two users with dominant resources as $r_i > r_i^{'}$ and both get the allocation based on MLF-DRS algorithm in 2 . As MLF-DRS allocates resources proportionally among the users based on their demand profiles, there is $\Pi(r_i) < \Pi(r_i^{'})$. In particular, if we assume two users i, j as the allocated tasks are determined based on the f^k; hence $r_i^k < f^k$, and $r_j^k > f^k$ (see lines 26 and 27). In particular, the allocation for user j is always greater than user i as $\Pi_j^k > \Pi_i^k$. Consequently, user i is unable to envy user j' allocation. Therefore, FFMRA meets envy-free property. □

Theorem 4.19. *FFMRA satisfies sharing-incentive.*

Proof. Given that there are two groups of tasks denoted by r, r^*, and FFMRA increases the allocation of dominant resources of all users in each group based on the maximum share by proportionality based on $\rho(d_i^k)$ and $\rho(\tilde{d_i^k})$ (see line 11 in algorithm 3). We need to guarantee that all tasks dominated on a specific resource type must be allocated by at-least $1/n$ of the resource pool. It is assumed that the entire resource pool is divided among two groups of dominant, and non-dominant resources based on line 18. Hence, if the total resources of the resource pool is denoted by $\tilde{R}$, then at-least $\tilde{R}/2$ of total resources is assigned to a group of users with dominant resources. Indeed, FFMRA sums up all dominant resources together and gives them the highest proportion of resource pool capacity. Therefore, by balancing the load in each specific resource, FFMRA ensures that each dominant resource gets at least $1/n$ of actual resource capacity. As an example (see 4.3.2.5) in a scenario with four users , $\rho(D)$ for dominant resource on RAM is 26.6. Hence, this is more than half of the RAM capacity. Accordingly, taking into account the final allocation based on MLF-DRS in 13, the allocation for each user in a specific queue is at least $1/n$. Therefore, it is guaranteed that FFMRA satisfies sharing incentive property. □

Theorem 4.20. *FFMRA satisfies Pareto-efficient.*

Proof. Again, assuming that r, r^* denote RAM and CPU intensive tasks respectively. Any requested task r, r^* is maximized in terms of allocation based on a specific dominant resource type without decreasing the allocation of other tasks. In other words, if there are two users i and j, using a saturated resource r, then increasing the dominant share of user i leads to a decrease in a dominant share of user j. However, in every step of the FFMRA algorithm, by increasing the dominant resource of a user, other user's dominant shares are maximized as well. To proof this, we need to first refer to the proportion of the entire resource pool for all dominant resource types which is indicated by $\rho(d_i^k)$ (see 9 in the algorithm 3). This parameter guarantees that in the first step, regardless of users' demand profiles, an equal proportion of resources is allocated to them. In particular, no user's allocation is maximized based on his/her entitlements. We already demonstrated that in the second hierarchy of FFMRA, the final allocation is determined by employing MLF-DRS policy (see algorithm 3, line 18). Hence, under MLF-DRS policy, the allocation for each user is maximised proportionally without affecting others' allocation based on $P = r_i^k.A_i^k / \sum_{i=1}^{n} r_i^k$. Therefore, we can confirm that the allocation under FFMRA is Pareto-efficient.

□

Theorem 4.21. *FFMRA meets strategy proof requirements.*

Proof. To proof the strategy-proof property under the FFMRA allocation policy, we assume a user with demand profiles r_i^k and $r_i'^k$ as the true and misreported requests respectively. Given that FFMRA increases dominant resources based on available resources in each stage, if a user with r_i^k tries to manipulate the server by $r_i'^k$ and considering that the capacity constraint is taken into account, in that case, the constraint could be violated by misreporting the true demand by any user. As FFMRA employs the MLF-DRS allocation mechanism, to prove the concept of strategy-proof, we investigate the final allocation for users in each group based on dominant and non-dominant resources based on line 18 in algorithm 3. In particular, FFMRA tries to allocate a proportion of entire resource pool to both of these groups, considering $\rho(d_i^k)$ and $\rho(\tilde{d}_i^k)$ (see lines 9 and 10, algorithm 3). Hence, at this level, there is no specific allocation for each user as all users are encapsulated in particular groups of dominant, and non-dominant resources based on line 5 and 6. Correspondingly, the allocation is determined by employing MLF-DRS and the new fair queuing mechanism, discussed in 4.2.2 and algorithm 1. Hence, the

strategy-proof under MLF-DRS 4.9 is applied to FFMRA and it is enough to prove that the allocation under FFMRA is Strategy-proof.

□

4.4 H-FFMRA

In this section, we propose a new multi-resource fair resource allocation algorithm called H-FFMRA in heterogeneous servers. H-FFMRA inherits all basic allocation principles associated with FFMRA. Accordingly, in section 4.4.1 we investigate the main motivation behind H-FFMRA. Then, section 4.4.4 introduces the formulation of H-FFMRA in multi-server settings, supported by examples. The fairness measure is also proposed in section 4.4.5. Finally, we evaluate the fairness features satisfied under H-FFMRA in 4.4.6.

4.4.1 Motivation

In FFMRA, the main focus was on a single server, whereas modern data centers are composed of multiple servers with distinct configurations. Applying both DRF, and FFMRA in a naive extension form, and separately across all servers violates Pareto-efficiency corresponding to DRF and intuitive fairness in FFMRA. To have a better comprehension of the problem, we refer to the example in Figure 4.9.

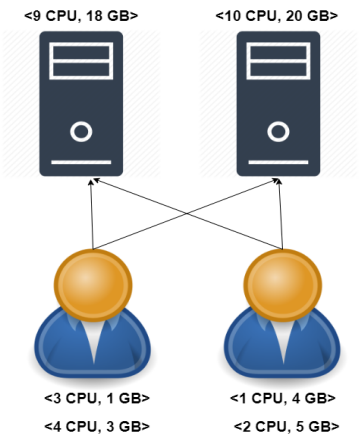


FIGURE 4.9: An example with two users submitting their tasks over two servers

Assume there are two users lets say 1, and 2 with demand vectors $(1CPU, 4GB)$, $(2CPU, 5GB)$, and $(3CPU, 1GB)$, $(4CPU, 3GB)$ respectively. Also, both users are eligible to run tasks in servers 1, and 2 with the capacity vectors $(9CPU, 18GB)$, and $(10CPU, 20GB)$. We assume that the DRF is separately applied in both servers. Therefore, user 1 receives $(3CPU, 12GB)$, and $(4.48CPU, 1.2GB)$ in server 2. Also, user 2 schedules, $(6CPU, 2GB)$, and $(5.52CPU, 4.4GB)$ in servers 1, and 2 sequentially. Hence,

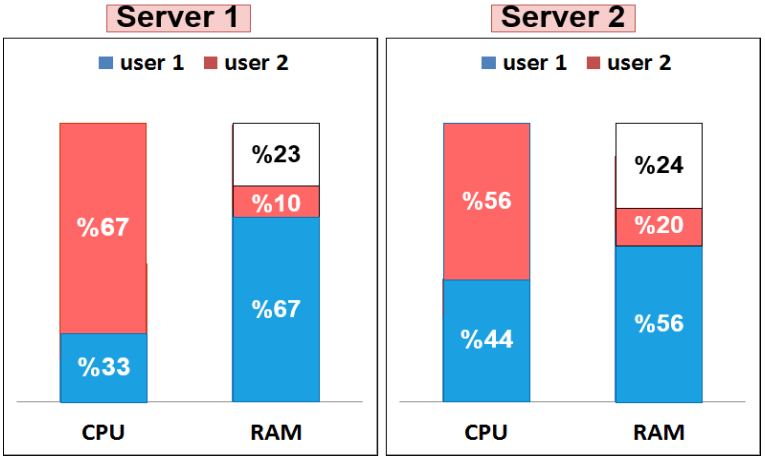


FIGURE 4.10: The allocation of resources in DRF over two servers

based on Figure 4.10, an inefficient allocation occurs in both servers that are about 23% of wasted resources. Moreover, the distribution of resources among users based on the dominant and non-dominant shares are not in the same percentage. The main assumption is to equalize all shares across multiple servers to achieve a trade-off between fairness and efficiency and intuitively fair resource allocation. Correspondingly, based on the same scenario in 4.9, users 1, and 2 benefit 69% of the entire resource pool in each server for dominant shares plus 31% equal percentage in all servers with regards to non-dominant shares(see Figure 4.11).

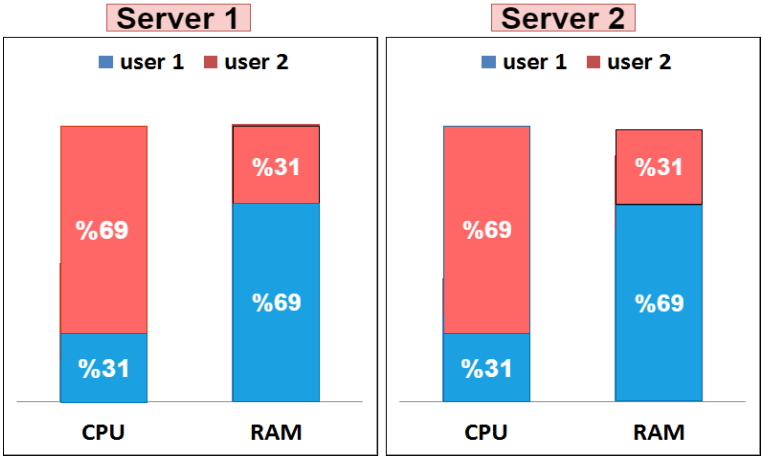


FIGURE 4.11: The allocation under H-FFMRA which indicates that resources are evenly allocated over two servers

4.4.2 Fair allocation properties in a multi-server setting

- ***Pareto-efficiency***: Given that there is n number of users in which $i \in U$, then, the number of tasks allocated to each user i with a demand profile r_i^k cannot be

increased without decreasing the allocation of other users $i^{'}$, $i \neq U$ (Tahir et al. 2015).

- ***Envy-free***: A user is unable to envy another user's allocations:

$$\Psi d_{is}^{k}(\Pi_{i}^{k}) \geq \Psi d_{is}^{k}(\Pi_{i^*}^{k}), \forall i, i^* \in U, i \neq i^* \tag{4.32}$$

Where $\Psi d_{is}^{k}(\Pi_{i}^{k})$ and $\Psi d_{is}^{k}(\Pi_{i^*}^{k})$, indicate the number of tasks dominated on a particular resource type that users i and i^* can schedule respectively based on dominant resources.

- ***Sharing-incentive***: This property confirm that no user is able to better-off others if all resources are divided fairly among them (Dandachi 2015):

$$\Psi d_{is}^{k}(\Pi_{i}^{k}) \geq \Psi d_{is}^{k}(c_{s}^{k}/n), \forall i \in U. \tag{4.33}$$

The inequality in 4.33, indicates that the sharing-incentive is achieved while the number of allocated tasks, dominated on a specific resource type must be equal or greater than the fair-share in each server.

- ***Strategy-proof***: Any user i is unable to increase his/her allocation by misreporting demands:

$$\Psi r_{is}^{'\,k} \leq \Psi r_{is}^{k}, \forall i \in U, \forall r_{is}^{k\,'} \neq r_{is}^{k}. \tag{4.34}$$

4.4.3 System setting

Assume that there are S heterogeneous servers represented by $S = (1, 2, ..., s)$ with the capacity vector $C = (c_{s}^{1}, ..., c_{s}^{k})$. Also there are n users indicated by $U = (1, 2, ..., n)$ compete over k resource types as $R = (1, 2, ..., k)$ with resource request profiles, presented by $X = (r_{i1}^{k}, ..., r_{is}^{k})$, where all demands are always positive for each user. i.e, $r_{is}^{k} > 0, i \in U$.

Definition 4.22. Consider resource k as the heaviest and non-heaviest demand of submission r by user i over the maximum capacity C of any server s. Hence, the dominant,

and non-dominant resources for each user specified by vectors $D = (d_{i1}^k, ..., d_{is}^k)$, and $\tilde{D} = (\tilde{d_{i1}^k}, ..., \tilde{d_{is}^k})$ respectively, are determined as follows:

$$d_{is}^k = max\frac{r_{is}^k}{C_s^k} \tag{4.35}$$

$$\tilde{d_{is}^k} = min\frac{r_{is}^k}{C_s^k} \tag{4.36}$$

Definition 4.23. If the allocation Π for each user i,$(1 \leq i \leq n)$ in server s is specified by $\Pi = (\Pi_{i1}^k, ..., \Pi_{is}^k)$, with the utility U_{is}^k, then, the maximum number of tasks that a user can schedule in each server is the minimum allocation of a specific resource type k (Liu, Zhang and Li 2017). Hence:

$$\Psi(U_{is}^k) = min\frac{\Pi_{is}^k}{r_{is}^k}, U_{is}^k = max(\Pi_{is}^k) \implies \forall i \in U, \exists r_{is}^k > 0. \tag{4.37}$$

Where, $\Psi(U_{is}^k)$ is the number of tasks allocated to user i in a particular server s.

Definition 4.24. A feasible allocation Π exists if the total allocation in each server is not greater than the total capacity of resources according to the following inequalities.

$$\sum_{i=1}^{n} \Pi_{is}^k \leq C_s^k, \forall i \in U \tag{4.38}$$

$$\sum_{i=1}^{n} r_{is}^k \leq C_s^k, \forall i \in U \tag{4.39}$$

Definition 4.25. The allocation Π is non-wasteful, if any user i schedules at least one task in each particular server s. Hence:

$$\sum_{i=1}^{n} \Psi(U_{is}^k) \leq C_s^k, \forall r_{is}^k, i \in U, s \in S. \tag{4.40}$$

Accordingly, for any allocation Π_{is}^k, and $\Pi_{is}^{*\,k}$, the allocation meets non-wasteful allocation if the following condition is satisfied.

$$\Psi_i(\Pi_{is}^{*\,k}) < \Psi_i(\Pi_{is}^{k}) \tag{4.41}$$

4.4.4 H-FFMRA

The recent developments fall short of satisfying intuitive fairness as Pareto-efficiency is not achieved exclusively for a specific number of users. The consequences of such allocation may severely hit those users with an identical type of dominant resources across multiple servers. In particular, if a user schedules a large number of dominated CPU tasks, then the probability of receiving the least amount of that resource is considerably high. For example, in some conditions, a user with a dominant resource on CPU demanding 5 units of CPU may get the same amount of that resource type. However, another user with the memory-intensive task may occupy the most proportion of the resource pool. It leads to a highly imbalanced allocation that noticeably affects the fairness and efficiency trade-off to sharing multiple resources.

To address the above-mentioned issues, in this section a fully fair allocation mechanism called H-FFMRA is proposed in multiple server profiles. H-FFMRA is the generalization of FFMRA in multiple servers. It captures the dominant, and non-dominant resources concerning each server, and performs β-fairness to evaluate how resources are evenly distributed among users over multiple servers. A two-level hierarchy approach in FFMRA is also applied to each server. Compared to the DRFH mechanism that uses the global dominant shares, in H-FFMRA we introduce the global aggregate dominant and non-dominant resources to achieve a fully fair resource allocation and a trade-off between fairness and efficiency. Accordingly, the H-FFMRA guarantees that a group of users with tasks dominated in a particular resource type, get a desirable share of resources based on their demands.

4.4.4.1 Problem formulation

The allocations under H-FFRMA are determined, using the Global Aggregate Resource (GAR), considering both dominant and non-dominant resources. Generally speaking, GAR aims to sum up all dominant and non-dominant resources across all servers.

$$GAR(d_{is}^k) = \sum_{i=1}^{n} d_{is}^k, \forall r_{is}^k \implies r_{is}^k \in D \tag{4.42}$$

$$GAR(\tilde{d}_{is}^k) = \sum_{i=1}^{n} \tilde{d}_{is}^k, \forall r_{is}^k \implies r_{is}^k \in \tilde{D} \tag{4.43}$$

Definition 4.26. An allocation satisfies H-FFMRA, if for all servers($s \in S$), there is at least one user with various submissions in which $(r_{is}^k) > 0$. Furthermore, H-FFMRA is feasible if any allocation Π corresponding to user i cannot be maximized without decreasing the allocation for any user j. i.e, $\Pi_{is}^k \geq \Pi_{js}^k$.

Definition 4.27. The maximum allocation that a user receives from a possible number of servers is measured by a utility specified by $U(\Pi_{is}^k)$.

$$\sum U(\Pi_{is}^k) \leq C_s^k, \forall i \in U. \tag{4.44}$$

This is worth mentioning that a user may receive the allocation from multiple servers.

4.4.4.2 H-FFMRA allocation

According to Algorithm 4, in the first level of the hierarchy, the proportion of the total resource capacity of all servers, indicated by $\rho(d_{i,T_s}^k)$, and $\rho(\tilde{d}_{i,T_s}^k)$ is calculated for dominant, and non-dominant resources in entire system. T_s which stands for total servers is determined for all users, sharing the entire resource pool with dominant, and non-dominant resources. Secondly, the proportion of $\rho(d_{i,T_s}^k)$, and $\rho(\tilde{d}_{i,T_s}^k)$ are specified for each server with respect to dominant and non-dominant resources, represented by φ_{d_s}, and $\varphi_{\tilde{d}_s}$, respectively. If $G_{d,s}^k$ and $G_{\tilde{d},s}^k$ denote a group of users with specific dominant, and non-dominant resources sequentially in each server. Then, the proportion of φ_d and $\varphi_{\tilde{d}}$ are used to calculate final shares, employing MLF-DRS 4.2.1 based on the following maximization problem. Like FFMRA, this process guarantees that the trade-off between the fairness and efficiency is achieved as a correlation between dominant and non-dominant resources is maintained across multiple servers.

$$
\begin{aligned}
\text{maximize} \quad & (r_{i1}^k, ..., r_{is}^k) \\
\text{subject to} \quad & \varphi_{d,s} \leq \rho_{d,s} \\
& \varphi_{\tilde{d},s} \leq \rho_{\tilde{d},s} \\
& \rho(G_{d,s}^k) \leq \varphi_{d,s} \\
& \rho(G_{\tilde{d},s}^k) \leq \varphi_{\tilde{d},s} \\
& \sum d_{is}^k + \sum \tilde{d}_{is}^k \leq C_s^k
\end{aligned}
\tag{4.45}
$$

Definition 4.28. An allocation under H-FFMRA is limited only to requested resources if any resource shortage happens in the system due to the low number of resources that are provisioned to all servers. Hence, the allocation is determined as follows:

$$\Pi(d_{is}^k) = r_{is}^k, \Pi(\tilde{d}_{is}^k) = r_{is}^k \tag{4.46}$$

4.4.4.3 An example

We refer to the example shown in Figure 4.9 where there are two users, submitting tasks with different demand vectors over two heterogeneous servers. As it was discussed earlier in section 4.4.1, user 1 has dominant resources on CPU, while user 2 has submission, dominated on memory. Therefore, based on the example, and according to Algorithm 4, H-FFMRA calculates $T_{c,s}$ for all dominant, and non-dominant resources in the entire resource pool, since $GAR(d_{is}^k) = 5 + 4 + 4 + 3 = 16$, and $GAR(\tilde{d}_{i,}^k) = 1 + 1 + 2 + 3 = 7$. As, $T_{c,s} = 10 + 20 + 18 + 9 = 57$, then $\rho(d_{i,T_s}^k) = (16 * 57)/23 = 39.65$, and $\rho(\tilde{d}_{i,T_s}^k) = (7*57)/23 = 17.35$. Accordingly, $\varphi_{d_{s1}} = (39.65*27)/57 = 18.78$, $\varphi_{d_{s2}} = (39.65*30)/57 = 20.87$, $\varphi_{\tilde{d}_{s1}} = (17.35 * 27)/57 = 8.21$, $\varphi_{\tilde{d}_{s2}} = (17.35 * 30)/57 = 9.13$. Then, $G_{d^{s1}}^{cpu} = (18.78 * 9)/27 = 6.24$, $G_{\tilde{d},s1}^{cpu} = (8.21 * 9)/27 = 2.73$, $G_{d^{s1}}^{ram} = (18.78 * 18)/27 = 12.52$, $G_{\tilde{d},s1}^{ram} = (8.21 * 18)/27 = 5.48$.For server 2, $G_{d,s2}^{cpu} = (20.87 * 10)/30 = 6.95$, $G_{\tilde{d},s2}^{cpu} = (9.13 * 10)/30 = 3.05$, $G_{d,s2}^{ram} = (20.87 * 20)/30 = 13.92$, $G_{\tilde{d},s2}^{ram} = (9.13 * 20)/30 = 6.08$. Figure 4.11, illustrates how resources are evenly shared among users. As can be seen in the figure, all users in the entire system with dominant resources on both CPU and RAM obtain 69% of resources. On the other hand, all users with non-dominant resources get exactly 31% of resources. This is against DRF policy as there is no balanced resource distribution. As a result, there is considerable resource wastage in both servers. This example indicates that a fully fair resource allocation and the trade-off between fairness

and efficiency are achieved. The example also indicates that by increasing a user's allocation, others' allocations are increased as well. This satisfies the Pareto-efficiency and sharing-incentive features as all users with dominant shares, receive at least 1/n of any resource type. This indicates that the allocation under H-FFMRA is intuitively fair.

This is worth noting that as there is only one dominant resource in each specific resource type, the allocation is not determined by MLF-DRS. Accordingly, in large-scale scenarios, since there is more than one dominant resource of a particular resource type in each server, the final allocation can be calculated by employing the MLF-DRS algorithm.

Algorithm 4 H-FFMRA allocation

Input: $D, \tilde{D}, C$
Output: $\rho(G_{d,s}), \rho(G_{\tilde{d},s}), \Pi_i(d_i^k), \Pi_i(\tilde{d}_i^k)$

$D = (d_{i1}^k, ..., d_{is}^k)$
$\tilde{D} = (\tilde{d}_{i1}^k, ..., \tilde{d}_{is}^k)$
$R \leftarrow (1, 2, ..., k)$ Resource vector
$S \leftarrow (1, 2, ..., s)$ The vector contains all servers
$U \leftarrow (1, 2, ..., n)$ total users in the system
$X \leftarrow (r_{i1}^k, ..., r_{is}^k)$demand vector
$T_{c,s} \leftarrow \sum C_s^k$ Sum of total resources in entire system
$\sigma_d \leftarrow \sum d_{is}^k$ Sum of all dominant resources
$\sigma_{\tilde{d}} \leftarrow \sum \tilde{d}_{is}^k$ Sum of all non-dominant resources
$\rho_{d,s} \leftarrow (T_{c,s} * \sigma_d)/(\sigma_d + \sigma_{\tilde{d}})$ The proportion of resource pool for all dominant resources
$\rho_{\tilde{d},s} \leftarrow (T_{c,s} * \sigma_{\tilde{d}})/(\sigma_d + \sigma_{\tilde{d}})$ The proportion of resource pool for all non-dominant resources
for each s in S **do**
    $\varphi_{d,s} \leftarrow (\rho_{d,s} * C_s^k)/\sigma C_s^k$ The Proportion for each server in terms of dominant resources
    $\varphi_{\tilde{d},s} \leftarrow (\rho_{\tilde{d},s} * C_s^k)/\sigma C_s^k$ The Proportion for each server in terms of non-dominant resources
end for
for each k in s **do**
    $\rho(G_{d,s}) \leftarrow (\varphi_{d,s} * C_s^k)/\sum C_s^k$ The proportion for each group of users in each server, dominated in each specific resource type
    $\rho(G_{\tilde{d},s}) \leftarrow (\varphi_{\tilde{d},s} * C_s^k)/\sum C_s^k$ The proportion for each group of users in each server with regards to non-dominant resources
end for
for i in $G_{d,s}$ **do**
    Apply MLF-DRS
end for
for i in $G_{\tilde{d},s}$ **do**
    Apply MLF-DRS
end for

4.4.5 Fairness

In particular, H-FFMRA follows the concept of fairness in a two-hierarchy approach that is already existed in FFMRA. Accordingly, in the first hierarchy it performs $\beta - fairness$ to measure the fair distribution of resources among groups of users in each server. In the second hierarchy, it applies *Jain's index* to all allocations in each server. In terms of $\beta - fairness$, an algorithm is fair if the percentage of allocated tasks for dominant, and non-dominant shares in all servers is the same. In other words, the system resources are equally distributed if the difference of the proportion of shares in all servers tends to *zero*. Therefore, $\beta - fairness$ for dominant, and non-dominant shares can be written as follows:

$$\beta_d = \left| \left(\sum \left(\frac{\Pi_i(d_{i1}^k)}{C_s^k} \right) \right) - ... - \left(\sum \left(\frac{\Pi_i(d_{is}^k)}{C_s^k} \right) \right) \right| \in (0, 1) \tag{4.47}$$

$$\beta_{\tilde{d}} = \left| \left(\sum \left(\frac{\Pi_i(\tilde{d}_{i1}^k)}{C_s^k} \right) \right) - ... - \left(\sum \left(\frac{\Pi_i(\tilde{d}_{is}^k)}{C_s^k} \right) \right) \right| \in (0, 1) \tag{4.48}$$

The formulations 4.47, and 4.48 indicate that whether a group of users in each server receive an equal proportion of resources in the entire resource pool. To measure resource allocation with fairness concerning each user in each server, the *Jain's index* is applied based on 4.49, and 4.50 for dominant, and non-dominant shares.

$$J(d_{i1}^k, ..., d_{is}^k) = \frac{(\sum d_{is}^k)^2}{n \sum (d_{is}^k)^2} \tag{4.49}$$

$$J(\tilde{d}_{i1}^k, ..., \tilde{d}_{is}^k) = \frac{(\sum \tilde{d}_{is}^k)^2}{n \sum (\tilde{d}_{is}^k)^2} \tag{4.50}$$

4.4.6 Fairness properties satisfied by H-FFMRA

In this section, all possible fairness features satisfied by H-FFMRA allocation are investigated. It has been already discussed that under H-FFMRA, there is at least a bottleneck resource that all users share in all existing servers. We go through all fair allocation features in detail, and extend them in the presence of multiple servers. In

terms of sharing incentive, this significant feature is extended across multiple servers to ensure that all users with dominant resources get at least $1/n$ of resources. An allocation Π satisfies a robust sharing-incentive if a user can schedule more tasks under this allocation.

Theorem 4.29. *The allocation Π under H-FFMRA meets bottleneck fairness if for each user $i \in U$ in corresponding server s, there is at least one saturated resource. Therefore, if G denotes a group of users in each particular server s and $\Psi(\Pi_{is}^k)$ indicates total allocated tasks to all users, there is a bottleneck resource b_s, since:*

$$d_{is}^k > d_{\tilde{i}s}^k \implies r_{i,\tilde{b}_s} > 0. \tag{4.51}$$

Definition 4.30. Under H-FFMRA allocation, it is not possible to maximize a user's allocation with submission $r_{is}^k > 0$, without decreasing others' allocations.

Lemma 4.31. *The allocation Π for each user with regards to each server s is referred to a non-wasteful sharing, if for each user i, there is $\rho(T_{c,s})$ (see line 10 in algorithm 4) as the proportion of servers' total capacity, and also there is $\gamma > 0$ which is the distribution of resources based on the proportion of total capacity of all servers. Consequently:*

$$\gamma = \rho(T_{c,s})(\vartheta_G(T_{c,s})) > 0 \tag{4.52}$$

$$\vartheta_G(T_{c,s}) = (T_{c,s})_{G,s}(\sum r_{G,s}) \tag{4.53}$$

Then, under MLF-DRS policy, each user i in server s, receives allocation Π, depending on the fair-share in each corresponding server, $f_s^k = \frac{C_s^k}{n}$. Therefore, based on 4.45 and 4.46, cwe consider $\vartheta(T_{c,s})$ which indicates the distribution of resources in groups of users with respect to each server. For each user i under allocation Π, the value of γ is relaxed to each user in a specific group G in server s. Hence, it is required to show that:

$$\Pi_{is}^k = \gamma(r_{is}^k) \tag{4.54}$$

$$\gamma = \rho(T_{c,s})(\Pi_{is}^k), \exists f^k, r_{is}^k > 0, \gamma > 0. \tag{4.55}$$

It is worth mentioning that in 4.55, $\vartheta(T_{c,s})$ has been replaced with Π_{is}^{k}, showing that $\vartheta(T_{c,s})$ leads to the final allocation for users in each specific server under MLF-DRS policy (see line 20 and 22 in algorithm 4).

Proof. To proof Lemma 4.31, we refer to the modification of multiple server specification in DRFH (Wang et al. 2013), taking into account that $\Pi_{is}^{k} = \gamma(r_{is}^{k}) \implies i \in U, k \in R$, then:

$$\Pi_{is}^{k}/d_{is}^{k} = \gamma(r_{is}^{k})/d_{is}^{k} = \gamma(r_{is}^{k}) \sum d_{i,s}^{k} \tag{4.56}$$

Hence, based on 4.56, the overall distribution to each group G in server s can be written as follows:

$$\Psi(\Pi_{is}^{k}) = min(\Pi_{is}^{k}/\sum d_{is}^{k}) = \gamma(\sum d_{is}^{k}) \tag{4.57}$$

Therefore, according to 4.57, for any allocation $\tilde{\Pi_{is}^{k}} < \Pi_{is}^{k}$, and for any resource $\tilde{K}$, considering $\tilde{\Pi_{is}^{k^*}} < \Pi_{is}^{k^*}$, there is:

$$\begin{aligned} \Psi(\tilde{\Pi_{is}^{k}}) &= min(\tilde{\Pi_{is}^{k}}/d_{is}^{k}) \\ &\leq \tilde{\Pi_{is}^{k}}/d_{i}^{k^*} \\ &\leq \Pi_{is}^{k}/d_{i}^{k^*} = \Psi(\Pi_{is}^{k}) \end{aligned} \tag{4.58}$$

As a result, the inequality in 4.58 indicates that Π is non-wasteful under H-FFMRA.

□

Lemma 4.32. *H-FFMRA meets envy-free.*

Proof. As already discussed in 4.54, $\Pi_{is}^{k} = \gamma(r_{is}^{k})$. Therefore, if there are two users i, and j, it is necessary to prove that:

$$\Psi(\Pi_{is}^{k}) \geq \Psi(\Pi_{js}^{k}) \tag{4.59}$$

Therefore for each value, there is:

$$\gamma_j(\Pi_j^k) = \sum_{i=1}^{s} \gamma_i(\Pi_{is}^k) \tag{4.60}$$

Without loss of generality, the equation in 4.60 demonstrates that the allocated tasks to user j, based on the proportion of total capacity of servers γ is equivalent to the sum of exactly the same proportion, allocated to user i.

Accordingly, based on 4.60, the value of γ for user j could be a minimum value, taking into account 4.59. Hence, for user j, there is:

$$\gamma_i = \sum_{i=1}^{s} min(\gamma_j(r_{js}^k)/r_{is}^k) \tag{4.61}$$

Then, 4.61 yields:

$$\gamma_i \leq \sum \gamma_j = \gamma_i(\Pi_i^k) \tag{4.62}$$

Consequently, by considering 4.62, the allocation Π is *envy-free* under H-FFMRA.

□

Lemma 4.33. *H-FFMRA satisfies sharing-incentive property.*

Proof. To prove the sharing-incentive property, it is required to show that every user i with dominant resource, get at-least $1/n$ of resources as:

$$\Pi_i(d_{is}^k) = \vartheta_G(T_{c,s})/n \geq C_s^k/n \implies i \in G_s(d_{is}^k) \tag{4.63}$$

The equation in 4.63, indicates that the allocation for each user with the dominant resource in each group G in server s must be greater than, or equal to $1/n$ of server capacity. It is achievable if the distribution of resource pool in a group of users with dominant resources is also greater than, or equal to the *fair-share*. For example, if there are four users in a system with a capacity of 36, as a result, $f^k = 36/4 = 9$. It is possible, if there are two users with dominant resources, and $\vartheta_G(T_{c,s}) = 24$, and if for both users,

$d^k_{i,s} \leq f^k_s$. Then, both users get an equal share of 12. In this case, the sharing incentive is satisfied (see algorithm 2, line 13). Therefore:

$$\vartheta_G(T_{c,s})_{d^k_{is}} > \vartheta_G(T_{c,s})_{\tilde{d}^k_{is}} \implies d^k_{is} \leq f_k \tag{4.64}$$

Given the inequality in 4.64, and in a case if $d^k_{is} \geq f_k$ for at-least one of users, MLF-DRS maximizes user i's allocation by increasing the allocation for user j as well. Therefore:

$$\Pi^k_{js} \geq \vartheta_G(T_{c,s})/n \implies i \in (U, G_s(d^k_{js})) \tag{4.65}$$

Accordingly, 4.65 shows that the *sharing-incentive* is satisfied under H-FFMRA.

□

Lemma 4.34. *H-FFMRA fulfills Pareto-efficiency.*

Proof. First of all, it is required to start with the following equation:

$$\sum \vartheta_G(T_{c,s})_{d^k_{is}} + \sum \vartheta_G(T_{c,s})_{\tilde{d}^k_{is}} = T_{c,s} \tag{4.66}$$

Considering 4.66, in the first level, all system resources are distributed among all groups of users with dominant, and non-dominant resources (see algorithm 4, lines 13 and 14). Also, according to *non-wasteful* allocation, as f^k is a constant value; under MLF-DRS users receive resources based on f^k. Therefore, if there are two users i, and j $(i \neq j)$, the following conditions are applied.

$$\begin{cases} \Pi^k_{is} > \Pi^k_{js} \implies d^k_{is} > f^k, d^k_{js} < f^k, \\ \Pi^k_{is} = \Pi^k_{js} \implies d^k_{is}, d^k_{is} = f^k. \end{cases} \tag{4.67}$$

Consequently, by increasing the allocation for user i, the allocation for j is increased accordingly. Nonetheless, both users i, and j receive exactly the same allocations.

□

Lemma 4.35. *The allocation* Π *is strategy-proof under H-FFMRA*

Proof. Assume a demand vector denoted by $R = (r^k_{i1}, ..., r^k_{is})$ for any user i in all existing servers S. Moreover, consider another user $\tilde{i}$, misreporting demands with profile $R = (\tilde{r}^k_{i1}, ..., \tilde{r}^k_{is})$. For both users i, and $\tilde{i}$, the allocations are considered as $\Pi^k_{is} = \gamma(r^k_{is})$, and $\Pi^k_{\tilde{i}s} = \gamma(r^k_{\tilde{i}s})$ respectively, based on the proportion of the entire resource pool (see algorithm 4, lines 10 and 11).

Hence, if user $\tilde{i}$ misreports his/her demand over any specific server s with dominant resource $d^k_{\tilde{i}s}$, then the proportion of requested resources by user i is the minimum value v_i as:

$$v_i = min(r^k_{\tilde{i}s}/r^k_{is}) \tag{4.68}$$

Therefore:

$$\begin{aligned} \Pi^k_{\tilde{i}} &= \sum_{s=1}^{m} \gamma(\Pi^k_{\tilde{i}s}) \\ &= v_i \sum_{s=1}^{m} (\vartheta_G(T_{c,s})) \\ &= v_i(\tilde{\vartheta_G}(T_{c,s})) \leq \vartheta_G(T_{c,s}) = \Pi^k_{is} \end{aligned} \tag{4.69}$$

Based on 4.69, the value of γ for each user's allocation in server s equals all allocated tasks to the user across all servers. Then, the v_i ensures that the minimum aggregate proportion of the resource pool is allocated to the user. Consequently, the allocated tasks to another user considering $\tilde{\vartheta_G}(T_{c,s})$ is always less than the proportion of resources allocated to another user under $\vartheta_G(T_{c,s})$. Consequently, it is not possible for user $\tilde{i}$ to misreport demands. Therefore, according to 4.68, and 4.69, H-FFMRA meets *Strategy-proof* feature.

Assuming v_i in 4.70 for dominant resources, and without loss of generality, if d^k_{is} presents non-dominant resources, v_i could be written as follows:

$$v_i = min(r^k_{\tilde{i}s}/r^k_{is}) \leq \tilde{d}^k_{\tilde{i}s}/\tilde{d}^k_{is} \leq d^k_{\tilde{i}s}/d^k_{is} \leq 1. \tag{4.70}$$

□

4.5 A New Approach to Calculate Resource Limits with Fairness in Cloud Environments

In this section we introduce a new approach to assign resource limits with fairness (Hamzeh, Meacham and Khan 2019), considering the integration of fair resource allocation algorithms in the Kubernetes framework. Correspondingly, in 4.5.1, we investigate the main motivation behind the approach. Also, section 4.5.2 discusses the formulation of fair resource allocation algorithms based on Kubernetes infrastructure as well as the system design. Finally, in 4.5.4 we present a practical example of assigning resource limits with fairness in a small setting.

4.5.1 Motivation

While VMs provision resources in the IaaS layer, the virtualization in containers takes place in the operating system as multiple containers run on top of the OSKernal (Pahl 2015). The main purpose of containerization is to provide isolation between management and application development without concerning migration from one environment to another. Docker is one of the most popular containerization platforms that was introduced in 2013 to package applications with necessary dependencies. Within the Docker, codes are transformed easily to other environments (Singh and Singh 2016).

Nonetheless, by increasing the volume of demands, the management and coordination of containers require sophisticated developments. To overcome these problems, Kubernetes (Bernstein 2014) has been proposed in 2015 as a container orchestration platform to orchestrate different workloads corresponding to computing and networking operations. Moreover, Kubernetes presents different functionalities such as load balancing, deployment, and scaling of a wide range of workloads.

As fairness is recognized as being the most important dimension in cloud computing systems, however, to the best of our knowledge, it has not been having considered in Kubernete's environment. Although DRF has been considered in the Kube-batch project as a batch scheduler on top of the Kubernetes (kube batch n.d.), it is employed as a plugin just for registering some callbacks for actions such as a compare function to sort jobs and event handler.

The fairness issue in Kubernetes has only been considered in terms of resource quota. The resource quota is applicable when any number of users or teams share a cluster with a certain number of nodes. In this case, the resource quota prevents a team to use more than its fair share. Although this method is an approach to avoid an unfair situation in the Kubernetes, it is only for the limited number of cases not for a general situation. Furthermore, the resource quota and limits must follow dynamic settings which means that each pod must receive its share based on a specific demand profile. In particular, due to the heterogeneity of resources in Kubernetes, each pod consists of intensive resource requests over one of the resource types either in CPU or memory. Consequently, maintaining fairness among the pods could be a core criterion in which the maximization of allocations for each pod becomes an optimal solution.

We believe that setting up resource limits without considering fairness is not an appropriate solution in an environment where pods compete to get more resources. Accordingly, if the resource limits are not specified during the pod creation level, it could consume all resources in the corresponding node. Therefore, each pod should normally get a specific amount of resources to avoid starvation.

Taking into account the above-mentioned problems, in this section, we model and integrate fairness algorithms in Kubernetes infrastructure, trying to assign resource limits fairly among different pods running on a specific node. Indeed, we add new functionality to the Kube-scheduler to consider resource limits with fairness during scheduling decisions.

4.5.2 Fairness in Kubernetes

A submitted job in the Kubernetes is recognized as a pod. Each pod may contain one or more containers. At the pod creation level, resource limits and requests are assigned to each specific pod that is essential in resource allocation. However, resource allocation based on limits and requests is not sufficient to establish fairness in Kubernetes as existing pods may require the highest proportion of a particular resource type. Unfortunately, there is a lack of dynamic resource management in Kubernetes as users are required to specify resource limits before submitting a job. Moreover, the Kubernetes scheduler does not consider resource limits in scheduling time as it takes into account only the resource requests. Accordingly, if a pod tries to consume more than its resource

limit, it could be evicted by the kubelet and kernel. This is worth mentioning that it is possible to specify namespaces to limit resource usage of pods, however, according to the fairness feature that we already discussed in chapter 2, this may violate Pareto-efficiency, and sharing incentive properties.

The lack of resource limit management causes pod eviction at the node level as there is no fair mechanism incorporated with Kubernetes. Technically, setting each pod in a particular namespace and assign resource limits using a fair allocation policy such as DRF may prevent pods to consume more resources than their specified limits. The main intuition behind our proposed approach is to equalize dominant resources depending on their contribution within the corresponding node. Accordingly, the main aim is to maximize resource limits based on the DRF mechanism. While DRF is considered to add its features in Kubernetes, we also aim to define a model to integrate our proposed algorithms in the Kubernetes scheduler.

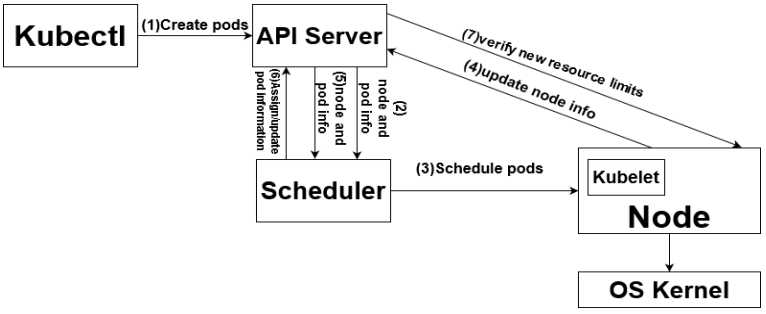


FIGURE 4.12: Pod processing in Kubernetes

This motivates us to add a capability to the scheduler to deal with resource limits. To do this which is illustrated in Figure 4.12, and Figure 4.13 we get all pod information, including resource limits, and requests, and the node information. Then a new resource limit is calculated, considering the dominant resources of all pods. After updating resource limits with the API server, the namespaces are assigned to each pod to avoid utilization above specified limits. In the optimization problem, the given constraint ensures that the sum of resource limits is less than or equals to the maximum capacity of each node. Consequently, the proposed mechanism prevents pod evictions and over-committed problems by identifying actual resource limits for all pods competing in a node to receive a desirable amount of resources.

Based on the example in Figure 4.14, there are two pods, each contains one container. Also, there is a node with a capacity of $(900m, 1800mi)$ of which m and mi denote

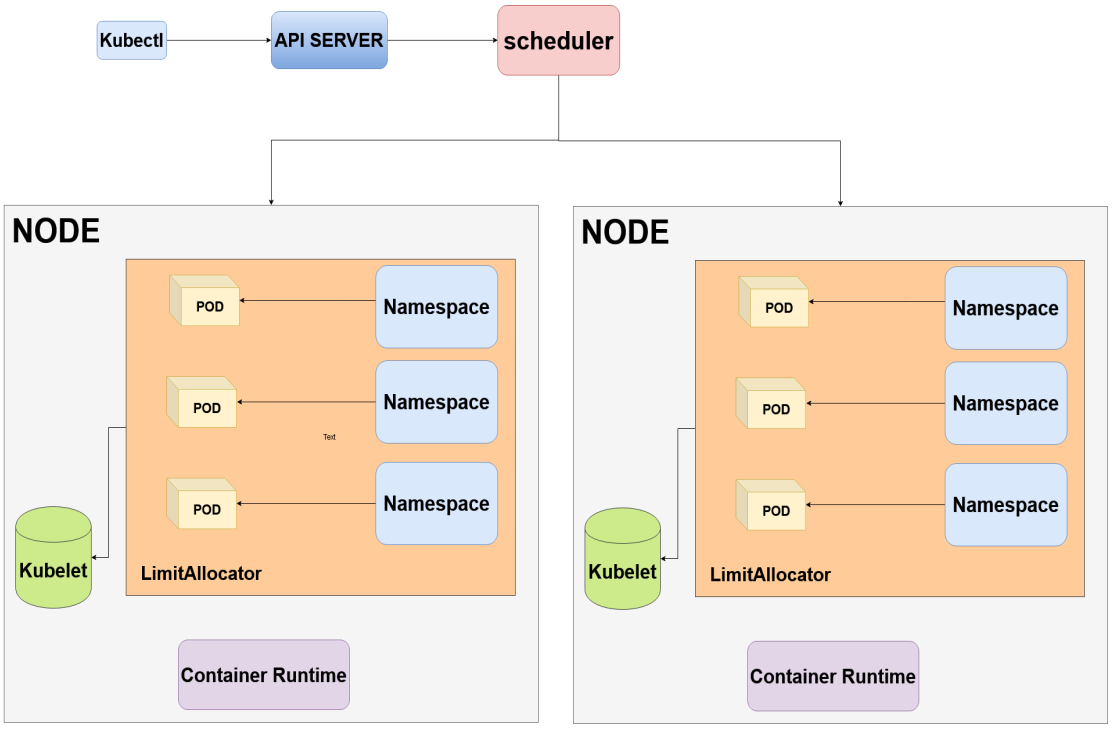


FIGURE 4.13: Using fair allocation and scheduling algorithms in Kubenretes - the figure illustrates each pod has been assigned with a namespace

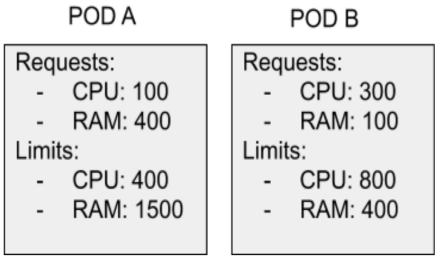


FIGURE 4.14: Pod configuration

CPU, and memory respectively. The Kubernetes attempts to allocate resources to each pod according to resource requests. Therefore, based on the default policy and what has been set up in resource requests and limits, $Pod1$ consumes up to $(200m, 650mi)$ of node capacity, and $Pod2$ utilizes a maximum $(400m, 300mi)$. Consequently, at least $(300m, 850mi)$ remain unused that could be allocated to other pods. For this reason, the Linux kernel allocates resources depending on defined resource limits. So, if DRF policy is applied to this specific example, memory in pod 1, and CPU in pod 2 are dominant resources. So, based on DRF, resource limits are determined as $(300m, 1200mi)$ for pod 1 and $(600m, 200mi)$ for pod 2. As a result, only $400mi$ is unused under the DRF policy. The main advantage of using fair allocation in the Kubernetes is that resources are shared fairly among the pods. Furthermore, users do not concern regarding pod

eviction, caused by OS Kernel since each pod is located in a specific namespace.

Generally, Our proposed model considers two different scenarios as follows:

1. **Resource limits are not specified**On this occasion, the proposed scheduler assigns actual resource limits based on requested resources. Generally, resource contention occurs if resource limits have not been specified. Although this problem is solved by grouping pods within namespaces, assigned limits are not complied with fairness criteria in multi-resource settings, leading to violate Pareto-efficiency, and sharing incentive properties.

2. **Resource limits are specified but aggregate limits exceed the node's allocatable resources**Accordingly, our mechanism takes node information and running pods within that node to recalculate resource limits with fairness to ensure the aggregate resource limits do not exceed the total capacity of that node. The example presented in Figure 4.14, two pods have been created by different resource requests and limits. If it is assumed that the total capacity of the node is $< 900mi, 1800m >$, and considering that the aggregate resource limits for CPU, and RAM equal 1200, and 1900. Hence, both CPU and RAM limits exceed capacity constraints. Consequently, one of those pods will be killed by the OS Kernel. We aim to solve this problem by recalculating resource limits by employing the DRF mechanism. DRF tries to maximize resource limits by equalizing them using the progressive filling algorithm.

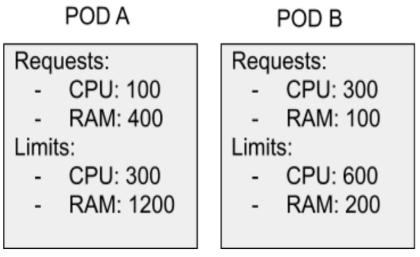


FIGURE 4.15: New resource limits assigned to pods A and B, considering DRf mechanism

As can be seen in Figure 4.15, new resource limits are determined for pods A and B based on DRF algorithms. This approach prevents resource contention and pod evictions while maintaining utility maximization, Pareto-efficiency, and sharing-incentive

in the Kubernetes environment. Moreover, to have more control over PODS, we define namespaces exclusively for each of those pods.

4.5.3 System Design

4.5.3.1 Mathematical implementation in Kubernetes

Primarily, in this section, a new approach is introduced to manage and orchestrate resource limits with fairness between pods. Typically, the main problem is represented as follows:

Consider there is a system with a set of users U and pods, denoted by P in a vector $|P| = p_1, p_2, \ldots, p_n$. Moreover, $|R| = 1, 2, ..., r$ presents a set of resources that each pod requests for those, as well as $|N| = 1, 2, ..., n$ a class of nodes, each of which could serve a certain number of pods. It is assumed that LP_r and xp_r refer to the resource limits and requests of a pod created by user U where the requested resources are subjected to be less than or equal to the resource limit as $(xp_r \leq lp_r)$. A pod that is scheduled in a particular node is represented by p_i, as a scheduled pod p in a corresponding node i. Dominant, and non-dominant resources for each pod with regards to the maximum capacity of each specific resource type indicated by C_{ir} are determined as follows:

$$dp_{ir} = max(xp_r/C_{ri}) \tag{4.71}$$

$$\tilde{dp}_{ri} = min(xp_r/c_{ri}) \tag{4.72}$$

The available resources in Kubernetes are called allocatable. Hence, without loss of generality, it could be determined as follows:

$$Allocatable = nodecapacity - (Kubereserved + systemreserved + evictionthreshold)$$

Hence, an allocatable resource, given to a specific pod i is specified as AL_{ri}. Hence, 4.71 and 4.72 could be changed to 4.73 and 4.74:

$$dp_{ir} = max(xp_r/Al_{ri}) \tag{4.73}$$

$$\tilde{dp}_{ri} = min(xp_r/Al_i) \tag{4.74}$$

In real-time scenarios, each resource r may be associated with a weight. Hence, if it is assumed that r has corresponding weight, denoted by f_{rp}, then 4.73 and 4.74 could be revised to 4.75, and 4.76 as follows:

$$d_{rp} = max(xp_r/f_{rp}) \tag{4.75}$$

$$nd_{rp} = min(xp_r/f_{rp}) \tag{4.76}$$

Given that $S(LP_r) = \sum Lp_r$ denotes the aggregate resource limits of a certain pod p, thus, the updated resource limits are calculated as follows:

$$\begin{aligned} &\text{maximize} && (p_1, p_2, ..., p_n) \\ &\text{subject to} && xp_r \leq C_{ir}. \\ & && s(lp_r) \leq C_{ir}. \end{aligned} \tag{4.77}$$

4.5.4 Practical example

As a preliminary implementation, a small case experiment is conducted in a single cluster using the minikube(minikube n.d.) with a maximum capacity of $(2000mi, 2000m)$.

Despite actual scenarios, it is assumed that the node information is known in advance. Therefore, the administrator assigns resource limits to pods regarding their demand profiles.

Figure 4.17 illustrates the configuration of pods, running in a single cluster as it is presented in Figure 4.16. Also, the total capacity of the cluster is $(2and2,038,624ki)$ of which 2 is the number of CPU cores equals 2000m and (2038624/1024=1990mi) is determined as the maximum capacity of memory. Nonetheless, the allocatable memory is less than the actual capacity of the corresponding node. Based on the mathematical implementations, the *frontend* and *frontend1* have requests dominated on CPU, while for *frontend2* and *frontend3* the requests are dominated on memory. According to the given policy, resource limits are evenly assigned to those applications with a dominant resource on CPU as both of which get approximately 36%, and 38% of the corresponding resource. On the other hand, for the last two applications with a dominant resource on memory, exactly 38% is considered for both. Whereas efficiency is a principle notion in

```
Addresses:
  InternalIP:  10.0.2.15
  Hostname:    minikube
Capacity:
 cpu:                2
 ephemeral-storage:  17784772Ki
 hugepages-2Mi:      0
 memory:             2038624Ki
 pods:               110
Allocatable:
 cpu:                2
 ephemeral-storage:  16390445849
 hugepages-2Mi:      0
 memory:             1936224Ki
 pods:               110
System Info:
 Machine ID:                 917e89bdbd5d488088a70c67754f529a
 System UUID:                B0E01513-E9D6-4453-9E0C-763B6E142F50
 Boot ID:                    54c6b824-1440-4eb7-8298-80e7204e206e
 Kernel Version:             4.15.0
 OS Image:                   Buildroot 2018.05
 Operating System:           linux
 Architecture:               amd64
 Container Runtime Version:  docker://18.6.2
 Kubelet Version:            v1.14.0
 Kube-Proxy Version:         v1.14.0
Non-terminated Pods:         (18 in total)
  Namespace                  Name                                CPU Requests  CPU Limits  Memory Requests  Memory Limits  AGE
  ---------                  ----                                ------------  ----------  ---------------  -------------  ---
  default                    frontend                            330m (16%)    720m (36%)  50Mi (2%)        81Mi (4%)      5d7h
  default                    frontend1                           555m (27%)    779m (38%)  150Mi (7%)       245Mi (12%)    5d7h
  default                    frontend2                           111m (5%)     159m (7%)   250Mi (13%)      736Mi (38%)    5d7h
  default                    frontend3                           222m (11%)    319m (15%)  350Mi (18%)      736Mi (38%)    5d7h
```

FIGURE 4.16: Allocated limits for pods using one cluster installed with Minikube - the output represents the actual resource limit assignment for pods considering cluster resources

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "50Mi"
        cpu: "330m"
      limits:
        memory: "81Mi"
        cpu: "720m"
```

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "150Mi"
        cpu: "555m"
      limits:
        memory: "245Mi"
        cpu: "779m"
```

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "250Mi"
        cpu: "111m"
      limits:
        memory: "736Mi"
        cpu: "159m"
```

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "350Mi"
        cpu: "222m"
      limits:
        memory: "736Mi"
        cpu: "319m"
```

FIGURE 4.17: Resource limits, calculated for all pods based on DRF policy

Kubernetes, we only take into account assigning resource limits with fairness between pods. The maximum amount of resources could be utilized by a pod up to the specified limit. However, a majority of resources could not be consumed by some pods and consequently remain unallocated. Technically, unused resources are reserved and then allocated to other pods in the queue.

The above-mentioned example illustrates how a fair allocation policy could be used in Kubernetes infrastructure. In chapter 4 we will show how other fair allocation and scheduling algorithms could be implemented in the Kubernetes scheduler.

4.6 Summary

In this chapter, three novel fair resource allocation algorithms in cloud computing systems are proposed. In MLF-DRS, we introduce a decision-making mechanism to allocate resources based on the boundaries of resource demands and fair-share. Besides, MLF-DRS is associated with a new queuing mechanism to prioritize tasks based on dominant resources. In FFMRA, we formulate two optimization problems to equalize the groups of users with dominant and non-dominant shares. Furthermore, to achieve a trade-off between fairness and efficiency, we propose a correlation among dominant, and non-dominant resources. Also, for measuring fairness in a multi-resource setting, we introduce a new evaluation formula. we then extend the applicability of FFMRA to the heterogeneous server profiles. In H-FFMRA, we present aggregate global resources for dominant and non-dominant shares to achieve a fully fair resource allocation. The fairness evaluations indicate that all these mechanisms satisfy desirable fairness features. Besides, a new model to assign resource limits with fairness in the Kubernetes framework was introduced to decrease the possibility of pod evictions.

Chapter 5

MRFS: A Multi Resource Fair Scheduling Algorithm in Cloud Computing

5.1 Introduction

Apart from the resource heterogeneity (Reiss et al. 2012), data centers are composed of different servers with distinct configurations in terms of resources. This diversity in servers and computational resources represents a considerable complexity in cluster management (Boutaba et al. 2012). Many extensions have proposed various ways to overcome shortcomings associated with DRF in heterogeneous servers such as (Wang et al. 2013; Tahir et al. 2015; Khamse-Ashari et al. 2017). These approaches have investigated alternative ways to schedule tasks based on the main intuition behind DRF. For example, the PS-DSF algorithm (Khamse-Ashari et al. 2017) has tried to schedule incoming tasks in the most efficient servers and in presence of placement constraints.

In particular, the missing point regarding the recent developments is that they have ignored how the number of dominant resources in a particular server may have an impact on fairness. In other words, equalizing the number of non-identical dominant resources in each server may increase the possibility of achieving Pareto-efficiency and sharing-incentive (RQ 1 and 7). In particular, satisfying these properties leads to an intuitively fair resource allocation. To address this issue, we propose a new policy-based fair task

scheduling algorithm namely MRFS (Hamzeh et al. 2020) to balance the submitted tasks concerning dominant resources, aiming to solve the Pareto-efficiency and sharing-incentive in a multi-server cloud setting. We believe that minimizing the competition between users with dominant resources may contribute toward maximizing the per-user utility function. Accordingly, we apply Lagrangian multipliers to the main optimization problem to reach the maximum optimal point, considering an equal number of non-identical dominant resources (RQ 7) and minimum aggregate dominant shares.

This chapter is organised as follows. In section 5.2 we discuss the motivation behind MRFS. Then, section 5.2.2 starts with the problem formulation along with examples. In continue, the section investigates the satisfaction of Pareto-efficiency and sharing-incentive properties under MRFS policy.

5.2 motivation

The existing approaches do not consider an equal assigning of tasks over multiple servers with regards to the number of dominant resources. In fact, equalizing the number of dominant resources in each server, and maintaining the minimum aggregate resource demands considering multiple resource types lead to maximizing users' utilities.

Let's draw an example to clarify the problem. Assume there are five users A, B, C, D, and E with demand vectors $(3CPU, 1GB), (5CPU, 3GB), (1CPU, 5GB), (2CPU, 7GB)$, and $(4CPU, 9GB)$ respectively. The main purpose is to schedule users' tasks in server s. This server is eligible to host a maximum of four users. Figure 5.1, presents a scheduling example of those tasks in a server from two perspectives. As can be seen one of $(5CPU, 3GB), (4CPU, 9GB)$ is eligible to be scheduled in the server according to Figure 5.1(a)(b). Otherwise, they could be scheduled in other servers if existed. Figure 5.1(a) refers to perfect scheduling as the number of dominant resources on CPU, and RAM(GB) is equal. However, the scenario in Figure 5.1(b) does not reach an optimal resource sharing as the number of tasks dominated on GB is more than CPU. Applying FFMRA, users C, and D receive a total of 26.6 ratios of the entire resource pool. Consequently, dividing it equally among them gives 13.4 units of GB. On the other hand, in a scenario represented in Figure 5.1(b), by increasing the population of tasks dominated on GB, the utilization is decreased as 30.66 ratio of resources is

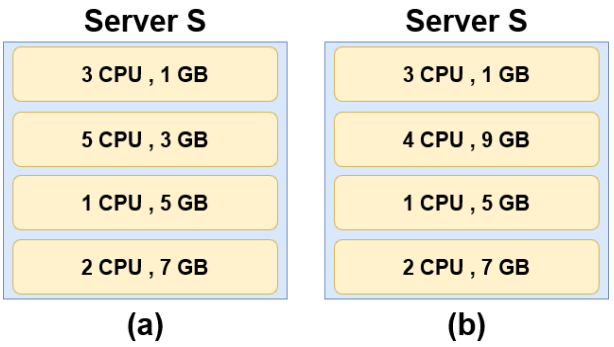


FIGURE 5.1: An example of scheduling scenarios

allocated to users C, D, and E. Furthermore, the aggregate demands dominated on GB is 21 in scenario (b) which is greater than the case in scenario (a) with 12 GB units. Therefore, imbalanced scheduling could adversely affect users' expectations in terms of utility maximization.

Unfortunately, the recent proposed approaches do not consider the competition based on the number of intensive tasks and also the minimum aggregate demands with dominant resources. This motivates us to design and develop a new policy-based scheduling algorithm to address these issues.

5.2.1 Basic resource scheduling setup

Resource scheduling is a primary management process in the IaaS delivery model. VMs in the cloud data center are recognized as scheduling units that are provisioned to heterogeneous resources. As it is shown in Figure 3.19, the scheduler regularly checks for the current status in the system to get available resources to check which resource is suitable to host users' tasks.

After resource provisioning, incoming jobs are placed in different queues. The main purpose of the resource scheduler is to find the most efficient node/server to host a set of cloudlets/tasks, waiting in those queues. The intuition behind resource scheduling in the cloud is to map a set of tasks $W = (W_1, W_2, ..., W_m)$ to available k types of resources $R = (1, 2, ..., k)$.

Let's assume that there is a system with heterogeneous servers and n number of users $U = (1, 2, ..., n)$ with demand profiles $X = (r^k_{i1}, r^k_{i2}, ..., r^k_{is})$. Also, assume that a user is eligible to submit any task including multiple types of resources. It is said that a user

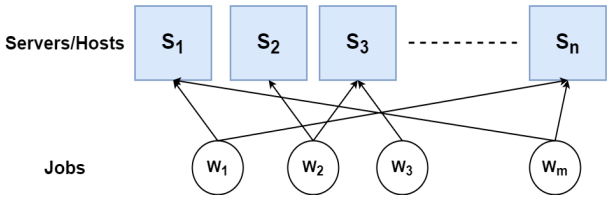


FIGURE 5.2: Scheduling tasks in different servers

submits a request dominated on a particular resource type if the following condition is satisfied:

$$d_{is}^{k} = max \frac{r_{is}^{k}}{C_{s}^{k}}, r > 0. \tag{5.1}$$

Where $d_{i,s}^{k}$ denotes a dominant resource that a user may request over any server s. Presumably, a submitted task by a user is determined locally as a task dominated on a particular resource type in the corresponding server. Respectively, the scheduler is responsible to map a task considering $d_{i,k}$ in a server to achieve the maximum utility for each user.

$$U_{i}^{max}(\Pi(d_{is}^{k})) \implies f : (f_{1s}, f_{2s}, ..., f_{is}) \tag{5.2}$$

The general utility maximization in 5.2 is subjected to find an optimal solution based on mapping function f, each binds cloudlets/tasks to the most efficient host. Typically, utility functions are the best indicators in evaluating the efficiency of any scheduling mechanism.

5.2.2 MRFS

In this section, a new policy-based fair task scheduling mechanism called MRFS is proposed in the heterogeneous cloud. MRFS considers dominant resources and performs a two-factor validation process to map a task to the most efficient server subjected to available resources. The first and the main factor is the frequency of dominant resources in each server, and the second one is the minimum aggregate dominant resources. If the last condition is not satisfied, the first factor is enough to schedule tasks. MRFS tries to maintain an equal distribution of tasks based on dominant resources on different

resource types. After submitting tasks, each task dominated on a specific resource type is placed in separate queues and then scheduled in the most appropriate server according to the specified rules. Technically, MRFS calculates dominant resources for each incoming task in all servers. If a server meets requirements and policies, then the task is scheduled in the corresponding server. It is important noting that the dominant resource of each task could be different from a server to server due to the diversity in terms of configurations. Hence, MRFS treats each task concerning its dominant resource type. On the other hand, if the configuration of all servers is identical, then MRFS considers a global dominant resource. In some conditions, if MRFS cannot find a suitable server to locate a task, the corresponding task is put into the non-dominant queue. However, if MRFS employs the FFMRA allocation policy, a dominant resource gets a fair-share of the resource pool as the highest proportion of the resource quota belongs to non-dominant resources. This is to make sure that those tasks get at least a fair-share of resources to satisfy the sharing-incentive property. However, in large-scale scenarios with thousands of servers, placing dominant resources in non-dominant queues happens very rarely.

Definition 5.1. Ψd_{is}^{k} refers to the number of dominant resources in each specific resource type k in each server.

Given that t indicates the time in a series $(t_0, t_1, ..., t_i - 1)$, the number of dominant resources on each particular resource type is updated in each time iteration t.

Definition 5.2. The available resources after occupying a server with several tasks is presented by A_s^k.

$$A_s^k(t) = C_s^k - \sum_{i=1}^{k} r_{is}^k \tag{5.3}$$

Where C_s^k is the maximum capacity of a resource type in each server, and r_{is}^k is the requested resource by user i in server s. Moreover, based on 5.3, and taking into account that in each iteration t, only one task is scheduled in each server, the available resources are updated accordingly.

Definition 5.3. A, and N as two sorted arrays, representing two vectors each of which indicates available resources, and the number of dominant resources respectively in each server at time t. Correspondingly, the first and the second items in the arrays are the

minimum values of vectors. After each iteration, these vectors are sorted by increasing the number of tasks in each server.

A server is indexed with a minimum aggregate dominant resource to 1, so that $(\sum r_{i1}^k \leq \sum r_{i2}^k \leq ... \leq \sum r_{is}^k)$. If $S = \{r_{i1}^k\}$, and $S^* = \{r_{i2}^k, ..., r_{is}^k\}$, then $\zeta(t) = \sum S^*/n - s$ denotes the average utilization of all servers except s_1 that has the minimum summation of requested resources at time t.

Definition 5.4. The parameter α is defined as an auxiliary penalty variable which maintains a correlation between the number of dominant resources with the current utilization at time t in server s. This is basically a penalty to show how the scheduler keeps the system in a desired state. To achieve α, it is assumed that $\Delta = \{\delta_1^s = \frac{\sum d_{i1}^k}{\Psi d_{i1}^k}, ..., \delta_s^k = \frac{\sum d_{is}^k}{\Psi d_{is}^k}\}$ indicates aggregate dominant resources over the number of dominant resource types k at time t. Then the values of Δ are normalised indicated by Λ with the maximum capacity of each particular resource type to capture α which is determined as follows:

$$\alpha = |\Lambda_1^k = \frac{\delta_1^k}{C_1^K} - ... - \Lambda_s^k = \frac{\delta^k}{C_s^k}|, 0 < \alpha \leq 1 \tag{5.4}$$

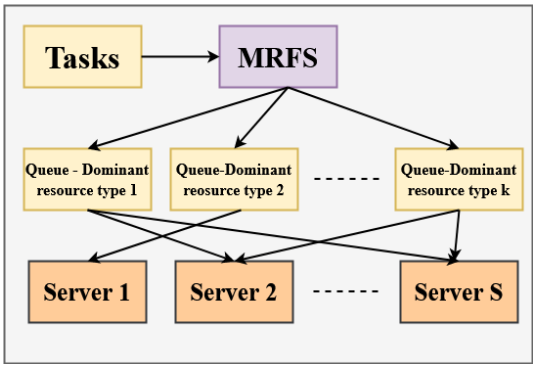


FIGURE 5.3: MRFS scheduler structure

Generally speaking, the value of α represents the maximum penalty if the number of dominant resources on a specific resource type is more than other types. In particular, if the number of dominant resources is the same, the value of α would be significantly high, while in a perfect condition, where $\alpha = 0$, there is no penalty at all. Therefore, the following conditions are satisfied to maximize per-user utility in a set of servers.

$$\begin{cases} \Psi(d_{i1}^k) = \Psi(d_{i2}^k) = ... = \Psi(d_{is}^k), \\ min \sum_{i=1}^{k} d_{is}^k(t). \end{cases} \tag{5.5}$$

Where for all resource types in server s, $min \sum_{i=1}^{k} d_{is}^k(t)$ denotes the minimum aggregate requested tasks dominated on k types of resources. This is worth mentioning that the above conditions capture a perfect mapping of tasks to servers. However, there are cases that at least one of those conditions is not fulfilled.

According to definitions 5.1,5.2,5.3 and 5.4, and given that Π_{is}^k is the allocated resource to user i in server S, we are ready to formulate the maximization problem as follows:

$$\begin{aligned} \max \quad & U_i(\Pi_{is}^k) \\ \text{subject to} \quad & \sum_{i=1}^{n} r_{is}^k \leq C_s^k \\ & S \leq \zeta \end{aligned} \tag{5.6}$$

The maximization problem in 5.6 has two constraints. To achieve a perfect optimal mapping, the second constraint is required to be satisfied. To solve this maximization problem and find the best server to schedule a task, we use *Lagrangian multipliers* (Wah and wu 2002). The Lagrangian multiplier is an approach to find the local maximum of a function $f(x_1, x_2, ..., x_n)$ in an optimization problem, subject to a set of equality or unequally constraints let's say $g(x_1, x_2, ..., x_n)$. The main intuition behind this method is to transform constraints to a set of *partial derivatives*. The derivatives of a function are applied to determine the critical points of that function. This is very useful to find a local maximum point. This is worth mentioning that MRFS employs a kind of placement constraint principles in (Wang et al. 2016; Khamse-Ashari et al. 2017). Accordingly, under MRFS, tasks get services from those servers that meet the conditions in Definition 5.4 and the maximization problem in 5.6.

As an example, a simple utility maximization problem is considered as follows (Kubilinskas 2008):

$$\begin{aligned} \max \quad & x = f(x) \\ \text{subject to} \quad & \mathrm{g}_i(x) \leq 0, \forall i = 1, ..., m \end{aligned} \tag{5.7}$$

Hence, it is possible to put the function f along with its constraint in a single maximization problem, using λ as a multiplier. Therefore, the problem in 5.7 can be written as follows:

$$x = max\mathcal{L}(x, \lambda) = max f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) \tag{5.8}$$

Solving 5.8 gives a local maximum value for a maximization problem in 5.7.

Accordingly, the optimization problem in 5.6 could be formulated with constraint in an integrated optimization problem using Lagrangian multipliers. As there are more than one constraints, the optimization problem can be written as follows:

$$\mathcal{L}(U, \lambda, \mu) = (U_i(\Pi_{is}^k) + \lambda[\, C_s^k - \textstyle\sum r_{is}^k] \,+ \mu[\, \zeta - S] \,) - \alpha \tag{5.9}$$

According to 5.9, The parameter U denotes the utility of a user i in any server s considering resource type k. Therefore, the optimization problem aims to maximize the allocation Π in a most efficient host. The second constraint is accompanied by α to keep the correlation between $\Psi(d_{is}^k)$, and $\sum d_{is}^k$.

To solve the Lagrangian function in 5.9, the First Order Necessary Condition(FOC)(Casas and Tröltzsch 2002) is applied. As can be seen in Figure 5.4 If U_i is the original user utility, U^* indicates the optimal one by solving the Lagrangian function. Accordingly, it is necessary to substitute U with U^* in FOC.

$$\frac{\mathcal{L}(U^*,\lambda^*,\mu^*)}{\partial U} = \left(\frac{\partial f}{\partial U}(U^*) - \lambda^* \frac{\partial \sum r_{is}^k(U^*)}{\partial U} - \mu^* \frac{\partial S}{\partial U}(U^*) \right) - \alpha = 0 \tag{5.10}$$

Consequently, the equality in 5.6 can be divided in the following inequalities:

$$\begin{cases} C_s^k - \sum r_{is}^k \geq 0, \lambda^* \geq 0, \lambda^*[\, C_s^k - \sum r_{is}^k] \,, \\ \zeta - S(U^*) \geq 0, \mu^* \geq 0, \mu^*[\, \zeta - S(U^*)] \,. \end{cases} \tag{5.11}$$

The maximization problem in 5.3 could be relaxed to one constraint in a case, where the second one is not required. Correspondingly, the problem can be written using a barrier function Φ as follows:

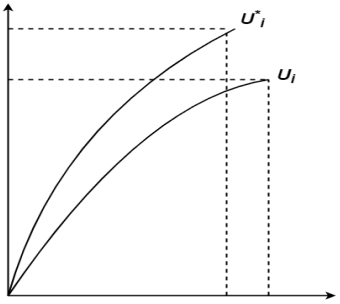


FIGURE 5.4: The optimal utility over the original utility

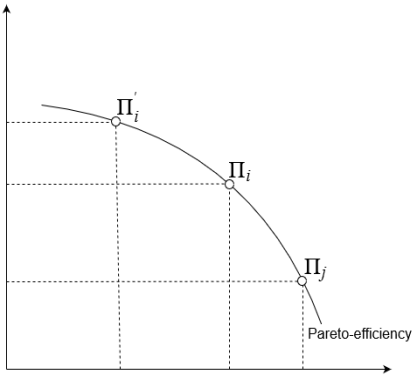


FIGURE 5.5: The curve represents Pareto-efficiency where $\Pi_i^{'}$ represents the improved Pareto-efficiency

$$maxU_i(\Pi_{is}^k) + \sum \Phi(d_{iS_1}^k) \tag{5.12}$$

The value of Φ could be selected as a logarithmic barrier function as follows:

$$\Phi(U_i(\Pi_{is}^k)) = -\sum log(d_{is}^k(U_i(\Pi_{is}^k))) \tag{5.13}$$

As MRFS performs H-FFMRA to calculate allocations, all fair allocation principles in FFMRA are exactly applied in MRFS. Among all fair allocation properties, we show that The sharing-incentive and Pareto-efficiency properties are improved under the MRFS scheduling mechanism.

Theorem 5.5. *there is Pareto-efficiency improvement under MRFS mechanism*

Proof. If there is a positive change on user i's allocation Π_{is}^k denoted by ${\Pi_{is}^{'}}^k$ so that ${\Pi_{is}^{'}}^k > \Pi_{is}^k$. Accordingly, user i is unable to worse-off user j'allocation under a perfect complementary utility function as $U_i(\Pi_i^{'}) \geq U_j(\Pi_j)$.

Algorithm 5 MRFS scheduling algorithm

Input: $r_{is}^k, d_{is}^k, C_s^k$
Output: $Q_{ds}^k, Q_{\tilde{d}s}^k$

$R \leftarrow (1, 2, ..., k)$ Resource vector
$S \leftarrow (1, 2, ..., s)$ The vector contains all servers
$U \leftarrow (1, 2, ..., n)$ total users in the system
$C = (c_1^k, ..., c_s^k)$
$X \leftarrow (r_{i1}^k, ..., r_{is}^k)$ demand vector
Q_{ds}^k Queue for dominant resource k in server s
$Q_{\tilde{d}s}^k$ Queue for non-dominant resource k in server s
$d_{is}^k = max\frac{r_{is}^k}{C^k}$ Dominant resource type k in server s
$s_d \leftarrow \sum d_{is}^k$
$t := 0$ time interval starts at 0
$\Psi(d_{is}^k)(t)$ number of dominant resources in each server at time t
$A \leftarrow C_s^k - \sum r_{is}^k$ Available resource in each iteration at t
for each s in S **do**
  if $(\Psi(d_{i1}^k) = ... = \Psi(d_{is}^k))$ & $s_d = min(\sum d_{is}^k)$ **then**
    $Q_{ds}^k \leftarrow r_{is}^k$ Placing requested task into dominants queue
  else if $(\Psi(d_{i1}^k) = ... = \Psi(d_{is}^k))$ **then**
    $Q_{ds}^k \leftarrow r_{is}^k$ Placing requested task into dominants queue
  else if $(\Psi(d_{i1}^k) = ... = \Psi(d_{is}^k)) + 1$ **then**
    $Q_{\tilde{d}s}^k \leftarrow r_{is}^k$
  else if $(\Psi(d_{is}^k) > \Psi(d_{is}^k)) + 1$ **then**
    $Q_{\tilde{d}s}^k \leftarrow r_{is}^k$
  end if
  $UpdateA$
  $t := t + 1$
end for

As it is shown in Figure 5.5, all allocations are in line with Pareto-efficiency. Hence, Π_i' is increased without affecting Π_j' allocation. The further improvement in terms of Pareto-efficiency is not feasible if an allocation reaches the desired state. Therefore, we need to show that under MRFS, decreasing the population of tasks dominated on a specific resource type reduces the competition among users, which leads to improving the Pareto-efficiency.

To show that Pareto-efficiency is improved under MRFS, we assume that a user i is given by a weight ω_i. Accordingly, for each allocation Π there is a social-welfare indicator (Negishi 2006) let's say S_w based on weighted aggregate utility as follows:

$$s_\omega(\Pi) = \sum_{i=1}^{n} \omega_i \cdot U_i(\Pi_i) \tag{5.14}$$

Hence, the allocation Π_ω maximizes the social-welfare over other allocations as follows:

$$\Pi_\omega \in argmaxs_\omega(\Pi) \tag{5.15}$$

Considering 5.14, and 5.15 under MRFS scheduling policy, minimizing the number of dominant resources leads to maximizing the utility of each user in a specific server S as follows:

$$min\Psi(d_{is}^k) \implies maxU_i(\Pi) \tag{5.16}$$

Therefore, the social welfare is improved according to 5.16 as:

$$S_\omega(\Pi^*) > S_\omega(\Pi) \tag{5.17}$$

□

Theorem 5.6. *MRFS improves the sharing-incentive property*

Proof. In particular, in order to proof the sharing-incentive property, we need to show that:

$$U(\Pi_{is}^k) \geq U_j(\Pi_{js}^k) \tag{5.18}$$

Therefore, the utility of user i is greater than or equal to anther user j's utility. This condition is assumed to be satisfied under FFMRA mechanism, if $U(\Pi_{is}^k) = \Pi_{is}^k / \sum_j(\Pi_{js}^k)$.

Z_o is taken into account as a solution under FFMRA in absence of MRFS, and also Z_o^* in presence of MRFS. Typically, the number of dominant resource (see line 11, algorithm 5) of a specific resource type under solution Z_o is more than that in Z_o^*, since, $\Psi d_{is}^k(Z_o) \geq \Psi d_{is}^k(Z_o^*)$. Therefore, considering λ, and μ, the utility of the user i is increased as follows:

$$(U(\Pi_{is}^k) + \lambda[C_s^k - \sum r_{is}^k] + \mu[\zeta - d_{i,1}^k]U(\Pi_{is}^k)) - \alpha \tag{5.19}$$

Consequently, any allocation Π under solution Z_o^* captures a strong sharing-incentive property. Hence, the inequality in (3.84) is applied to the optimal solution Z_o^*.

□

5.3 Summary

In this chapter, a novel fair task scheduling mechanism called MRFS is introduced. Under MRFS, we target to equalize the number of non-identical dominant resources in each server. In particular, we intend to reduce the competition among these kinds of resources to address the existing issues in terms of sharing-incentive and Pareto-efficiency. We also formulate an optimization problem, considering the maximum availability and minimum aggregate shares as rigorous constraints. Accordingly, We apply the Lagrangian multipliers to map incoming tasks to the most efficient servers. Furthermore, the mathematical proofs show that the MRFS achieves an improved degree of sharing sharing-incentive and Pareto-efficiency features.

Chapter 6

Applicability of the proposed algorithms

6.1 Introduction

The understanding of how to properly implement any resource allocation and scheduling algorithm requires a comprehensive investigation of the simulation and production environments. In both cases, taking into account the model-based design gives a clear view of how to implement those algorithms. Accordingly, in this chapter, we explore a model-based approach to deploy our proposed algorithms. In section 6.2, the Cloudsim is comprehensively explored as a simulation framework that has been specially designed for cloud computing environments. Moreover, the implementation of proposed algorithms is presented through the software modeling diagrams. Section 6.3, represents the applicability of proposed algorithms in BT's infrastructure, considering the Kubernetes as a container orchestration framework. We first study the architecture of the Kubernetes as a whole. Then, similar to section 6.2, the integration of proposed algorithms in this framework is indicated using software modeling principles.

6.2 CloudSim

CloudSim (Buyya et al. 2009) is a Java-based simulation framework that was designed in the cloud System labs at the University of Melbourne to simulate and model the cloud

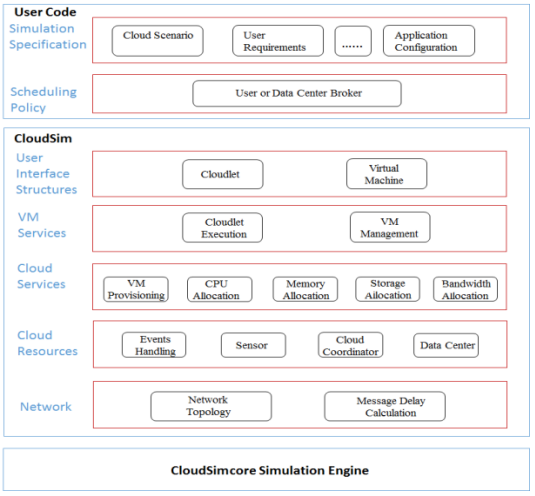


FIGURE 6.1: CloudSim architecture (Calheiros et al. 2009; Vahora and Patel 2015)

infrastructure. In the CloudSim all components such as Virtual Machines(VMs), data center, and resource provisioning policies are considered concerning their behaviors. Different resource provisioning techniques are easy to extend and develop. One can create and deploy own infrastructure and policies to manage a system and its resources. In other words, CloudSim is an interesting framework, allowing developers to investigate their proposed rules in an extensive and manageable environment without concerning real cloud infrastructure. A variety of classes are provided within the CloudSim to characterize different cloud components such as data center, users, resources, and VMs.

6.2.1 CloudSim Architecture

The whole architecture of the cloudsim is depicted in Figure 6.1. The simulation layer in the cloudsim comes up with a support to model and simulate the virtualized data center domains, comprising a set of interfaces to manage computational resources like CPU, memory, bandwidth, and disk.

This layer consists of operational actions such as running applications, monitoring the dynamic behavior of the system, and host provisioning to VMs. Allocation and resource provisioning policies are written in the simulation layer by changing and developing the provisioning of VMs. Hosts are provisioned to VMs where the applications are executed according to provided QoS parameters delivered by the SaaS layer. The user's code is the top layer in the cloudsim architecture that provides information about the application such as required resources for running a task as well as the information regarding hosts

such as, the number of VMs and resource allocation policies. Using this information and entities, a cloud developer may perform a mix of required actions such as creating and configuring workloads. He/she also may configure applications according to the pre-defined scenarios and testing the robustness of the system as well as implementing custom resource allocation, and scheduling policies.

ClouldSim is composed of a range of classes that demonstrate the CloudSim model. The data center is a fundamental component in the cloudsim where the IaaS is implemented and developed. Hosts that are assigned to VMs are managed by a datacenter entity using provisioning techniques specified by the Cloud provider such as VM provisioning, creation, and migration to maintain the life cycle of VMs. A VM is capable of hosting different applications subjected to constraints. In other words, one or more applications could be provisioned inside a single VM. Hosts provide computational resources to VMs like processing, memory, storage, and bandwidth. To manage VMs, hosts run the scheduler component. Scheduling in the ClouldSim is managed in two different ways: space-shared and time-shared policies. According to the space-shared scheduling policy, if there are many VMs, the next VM will not be started until the first VM finishes the execution period using a time slice. To calculate the finishing time of a VM, the following formulation is used (Buyya et al. 2009):

$$eft(p) = est + \frac{r_l}{capacity \times cores(p))} \tag{6.1}$$

where est(p) denotes the start time of a cloudlet(task) and r_l represents the total number of instructions that is required by a clouldlet to be executed within a specified VM. Subject to the availability and in a case that there are also idle cores to be assigned to VMs, the cloudlets are placed in the queue. So, the overall capacity of a host with regards to the Processing elements (PEs) is determined as follows:

$$capacity = \sum_{i=1}^{np} \frac{cap_i}{np} \tag{6.2}$$

where cap(i) represents the strength of processing of each specific element.

On the other hand, the time-shared policy which is implemented in *org.cloudbus.cloudsim.CloudletSchedulerTimeShared* allows multi-tasking which means that multiple cloudlets

can be executed inside a single VM. So, the overall processing capability of each individual host is determined as follows:

$$capacity = \frac{\sum_{i=1}^{np}(cap_i)}{max(\sum_{j=1}^{cloudlets} cores(j), np)} \tag{6.3}$$

In this thesis, we use the most useful methods within the Timeshare cloudlet scheduler class, that are shown in Listing 4.1 (CloudSim 2009).

```
getCurrentRequestedMips(): Gets the current requested mips.
getCurrentRequestedUtilizationOfBw(): Gets the current requested bw.
getCurrentRequestedUtilizationOfRam(): Gets the current requested ram.
getTotalUtilizationOfCpu(double time): Get utilization created by all cloudlets.
getTotalCurrentAllocatedMipsForCloudlet(ResCloudlet rcl, double time):
Gets the total current allocated mips for cloudlet.
```

LISTING 6.1: Cloudlet Scheduler timeshared class methods and functions

6.2.2 CloudSim classes

In this section we quickly overview different classes in the CloudSim.

1. *BwProvisioner:* This class introduces various techniques and models to allocate bandwidth as a resource to VMs. It is also responsible to allocate bandwidth among competing VMs in the whole data center. This class is extendable, so, one can employ his/her allocation policy. The bandwidth in this class can be reserved as much as needed until it reaches to the capacity constraint specified in the host.

   BwProvisioner has been implemented within *org.cloudbus.cloudsim.provisioners .BwProvisioner*. We use a number of important methods of this class to implement our proposed algorithms that are illustrated in Listing 4.2 (CloudSim 2009).

```
allocateBwForVm(Vm vm, long bw): Allocates BW for a given VM.
deallocateBwForVm(Vm vm): Releases BW used by a VM.
getAllocatedBwForVm(Vm vm): Gets the allocated BW for VM.
getAvailableBw(): Gets the available BW in the host.
getBw(): Gets the bw.
```

LISTING 6.2: Methods in BwProvisioner

2. *CloudCoordinator:* This class is an extension of data center which is responsible for monitoring the status of resources in the data center to apply dynamic decisions.

3. *CloudletScheduler:* Within this class, a set of scheduling policies(Time-shared and space-shared) are considered to calculate how processing power is provisioned to different VMs.

4. *Datacenter:* The aim of this class is to model the IaaS platforms, delivered by providers such as Google and Amazon. The DataCenter class has been developed in *org.cloudbus.cloudsim.Datacenter* which extends the *SimEntity* object, shown in Listing 4.3.

```
public class Datacenter extends SimEntity
```

LISTING 6.3: DataCenter Class

In this thesis, we take into account some important methods, implemented within the Datacenter object which is illustrated in Listing 4.4 (CloudSim 2009).

```
getHostList(): Returns the host list.
getCharacteristics(): Returns the characteristics.
getSchedulingInterval(): Returns the scheduling interval.
getVmAllocationPolicy(): Returns the vm allocation policy.
getVmList(): Returns the vm list.
```

LISTING 6.4: DataCenter object methods

5. *Data Center Broker* Data center broker which is also known as cloud broker is basically a bridge between SaaS and cloud providers to setup negotiations such as QoS requirements.

6. *RamProvisioner:*This class encompasses memory provisioning techniques. In order to deploy a VM within a specific host, it is necessary to check this class to make sure that there is available memory. RamProvisioner is also depicted in *org.cloudbus.cloudsim.provisioners.RamProvisioner*. The most important methods that we apply to implement our proposed algorithms are mentioned in Listing 4.6 (CloudSim 2009).

```
allocateRamForVm(Vm vm, int ram): Allocates RAM for a given VM.
deallocateRamForAllVms(): Releases BW used by a all VMs.
```

```
deallocateRamForVm(Vm vm); Releases BW used by a VM.
getAllocatedRamForVm(Vm vm): Returns the allocated RAM for VM.
getAvailableRam(): v the available RAM in the host.
getRam(): Returns the ram.
getUsedRam(): Returns the amount of used RAM in the host.
```

LISTING 6.5: Methos in RamProvisioner class

7. *Vm:* VM class is responsible to model VMs that could be scheduled in a particular host. Different features related to VMs are stored inside an element. Moreover, resource allocation and provisioning rules are also accessible through this element. A VM is created and defined by the VM class within *org.cloudbus.cloudsim.Vm* object which is shown in Listing 4.7. There are also a number of methods within the VM object that we consider them in this thesis that is depicted in the listing 4.6 (CloudSim 2009).

```
public class Vm extends Object

Vm(int id, int userId, double mips, int numberOfPes,
int ram, long bw, long size, String vmm, CloudletScheduler cloudletScheduler)
```

LISTING 6.6: VM class

```
getBw(): Gets the amount of bandwidth.
getCloudletScheduler(): Returns the vm scheduler.
getCurrentAllocatedBw(): Returns the current allocated bw.
getCurrentAllocatedMips(): Returns the current allocated mips.
getCurrentAllocatedRam(): Returns the current allocated ram.
getCurrentRequestedBw(): v the current requested bw.
getCurrentRequestedRam(): Returns the current requested ram.
getCurrentRequestedTotalMips(): Returns the current requested total mips.
getHost(): Returns the host.
getId(): Returns the id.
getMips(): Returns the mips.
getNumberOfPes(): Returns the number of pes.
getRam(): Returns the amount of ram.
```

LISTING 6.7: methods in the Vm class

8. *VmAllocation Policy:* This class finds the most appropriate host to deploy VMs based on available resources. This class is used to implement MRFS algorithm. The block definition diagram depicted in Figure 6.2 illustrates how to implement MRFS scheduling policy using different components and classes in the CloudSim.

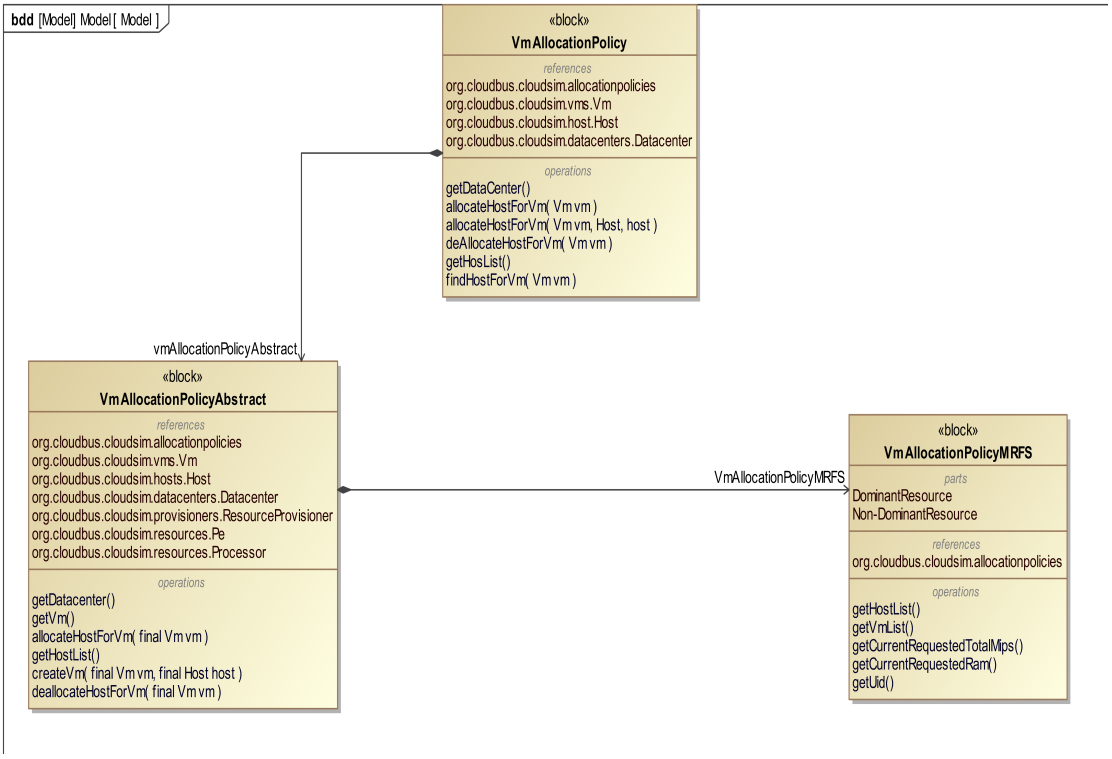


FIGURE 6.2: A block definition diagram that illustrates the technical implementation of MRFS algorithm and its applicability in the cloudsim framework.

Accordingly, two important classes *VmAllocationPolicy* and *VmAllocationPolicyAbstract* are essential to implement MRFS. Technically, it is necessary to extend *VMAllocationPolicyAbstract* and implement *VmAllocationPolicy*.

```
allocatePesForVm(Vm vm, List<Double> mipsShare): Allocates PEs for a VM.
deallocatePesForAllVms(): Releases PEs allocated to all the VMs.
getAllocatedMipsForVm(Vm vm): Returns the MIPS share of
each Pe that is allocated to a given VM.
getAvailableMips(): Gets the free mips.
getMaxAvailableMips(): Returns maximum available MIPS among all the PEs.
getTotalAllocatedMipsForVm(Vm vm): Gets the total allocated MIPS
for a VM over all the PEs.
```

LISTING 6.8: Methods implemented in the Vm Scheduler class

9. *VmScheduler:* The main purpose of this class that is implemented by the host component is to manage and implement provisioning policies either the space-shared or time-shared to allocate processing cores to VMs. VmScheduler is developed in *org.cloudbus.cloudsim.VmScheduler*, including different methods that are shown in the Listing 4.8.

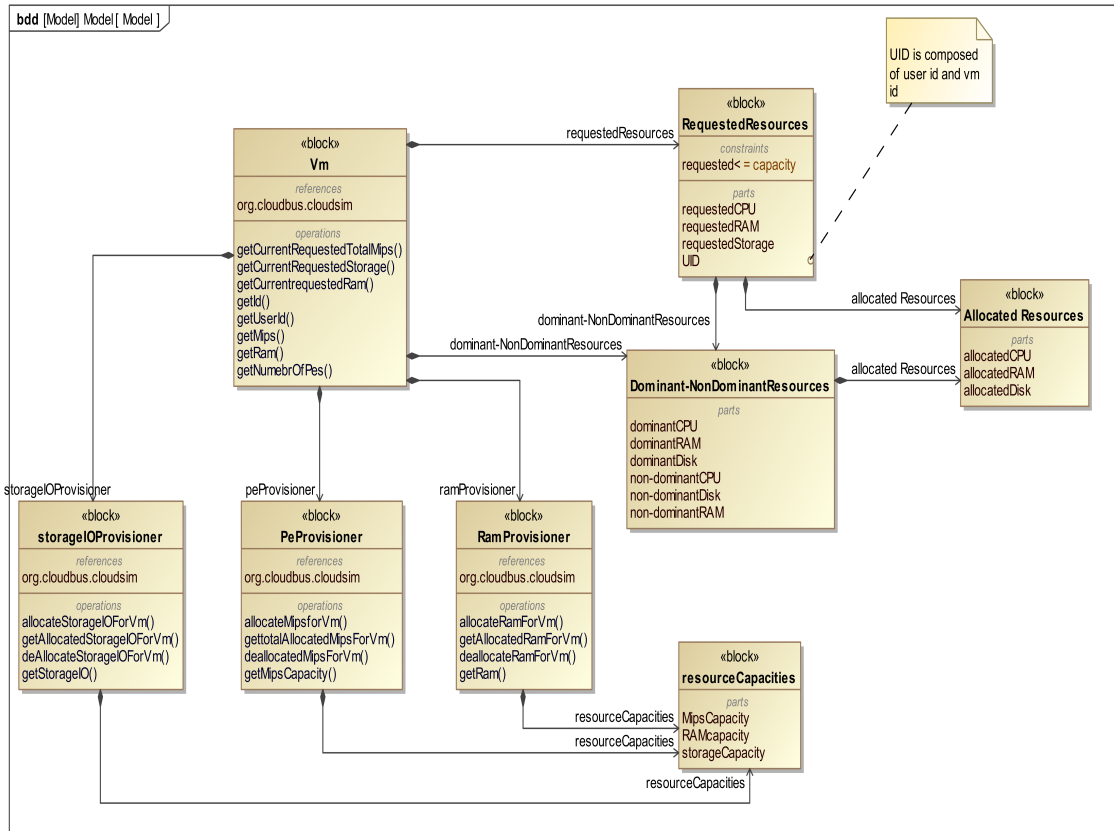


FIGURE 6.3: A low-level architecture of designing and implementing resource allocation algorithms in the CloudSim

6.2.3 Deploying proposed algorithms using CloudSim classes

The Figure 6.3, depicts different components in the CloudSim that are contributed in deploying our proposed algorithms. As can be seen in the figure, the core component is the *org.cloudbus.cloudsim* which consists of a variety of classes and methods. To do implementations, four important components are required such as, *StorageProvisioner, RamProvisioner, PeProvisiner and Vm.* In the first phase, *RequestedResource* object is created to determine requested resources using the *Vm* component as well as users, and VM ids. In the next step, *Resource Capacities* object gathers all resource capacity information from the *Storage, Ram and Pe* provisioner components. This is note that, the resources capacities are retrieved using the methods like *getStorageIO(), getMipsCapacity()*, and *getRam*. In order to calculate dominant and non-dominant resources, we consider *Vm* component and *resource Capacities* object. Accordingly, the *Allocated Resources* object gathers required information from the *VM* component, *Dominant and non-Dominant Resources*, and *Resource Capacities* objects. In the final step, allocated

resources in the *Allocated Resources* object are provisioned to all VMs as it is shown in the *Resource Provisioning to Vms* object.

6.3 The applicability in BT's infrastructure

British Telecom(BT) has a long experience in designing and developing Cloud-based applications. As a part of BT's project, we collaborate in designing resource allocation algorithms to be integrated to BT's cloud infrastructure. Therefore, we aim to deploy our proposed algorithms in the Kubernetes platform as a state-of-the-art cloud-based technology to provide fair, and efficient services to users. so, in this section, we extensively overview the Kubernetes architecture and its components that are essential in deploying proposed mechanisms.

6.3.1 Kubernetes

Kubernetes (K8s) is a well-known open-source platform for the container orchestration, automating deployment, scaling, and management of containerized applications. It gathers together all containers to keep up applications into a set of logical units to perform easy management and discovery operations (Hamzeh, Meacham and Khan 2019). A high-level architecture of the Kubernetes is presented in Figure 6.4. The Cluster is the most significant component in the Kubernetes which is composed of a bunch of running machines to manage containers.

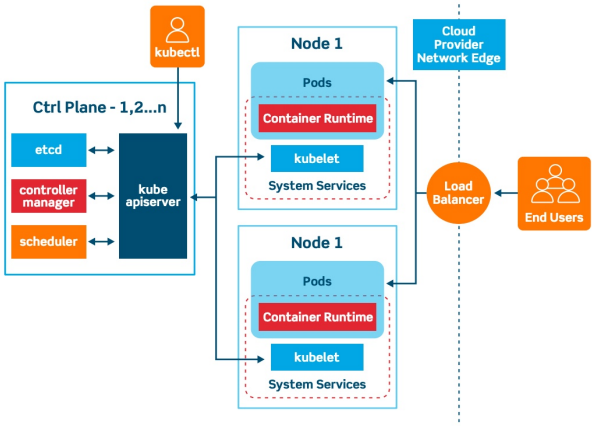


FIGURE 6.4: Kubernetes architecture (Pscaer 2018)

The Kubernetes mainly consists of two types of resources that are CPU and memory as computing resources for containers. Pods are considered the simplest and fundamental units in Kubernetes infrastructure. The scheduler in the Kubernetes scheduler is responsible for placing pods in available nodes. A node is referred to as a virtual machine, however, it is not created by Kubernetes. Instead, cloud providers such as Google Cloud, Amazon Web Services (AWS), and Microsoft Azure initiate nodes as virtual machines to host pods. The most important components within the Kubernetes cluster are listed as follows:

- Master: The master is considered as the main unit in Kubernetes architecture which provides various functionalities in order to manage, and monitor nodes and other components. The Master component is composed of the following parts.

1. Etcd: is key-value storage and backup agent in Kubernetes as all the information and configurations regarding the cluster are stored and accessible through the API server by worker nodes.

2. Kube-API server: The API server is a front-end object and a critical service within the master component that behaves like a bridge between different objects. The API is empowered using a JSON file. Furthermore, the *kubeconfige* package is responsible to handle communications within API. As a principle management component of the entire cluster, a user is allowed to configure Kubernetes workloads and organizational units. The API server is eligible to manipulate the state of different objects like pods, and services.

3. Kube-controller-manager: The shared state of a cluster is monitored by a controller via the API server to change the current state to an optimal point. The existing controllers in the Kubernetes are endpoints, replication, namespace, and service accounts. To reduce the complexity, all these controllers are integrated and compiled in a single binary.

4. Kube-scheduler: This component is responsible for deploying pods and services in suitable nodes. Different parameters are considered during the scheduling process such as resource requirements and limits, the quality of services of pods, affinity, and anti-affinity. The main function of a node is to check the requested resources

of all pods to make sure that the requested amount does not exceed the total capacity of the corresponding node.

- Node: A node is a worker machine, managed by the master component which is created by cloud providers to run containers. It also consists of different objects as follows:

1. Kubelet: Is one of the main components in the kubernetes architecture that places into each worker node to run all pods and checks them regularly to ensure that they work properly. The Kubelet also runs a set of health checks for all running pods. Then, it interacts with API server to report the state of a node and all pods running inside it in a certain intervals.
2. Kube proxy: As an object running in each worker node, kube proxy checks regularly the changes in all pods and services to keep the network up to date.
3. Container runtime: This object is placed at the lowest layer of a node to start and stop all pods and services. Docker is the most well-known container run-time.

6.3.2 Pod

A pod is referred to as the smallest element which is deployable inside the Kubernetes cluster. A single instance of an application is represented using a pod. A pod maintains one, or more containers that are mainly Docker containers. All containers within a pod are a single object that shares all resources, belonging to a given pod. Network and storage are two important shared resources for all created pods. In terms of networking, each pod is distinguished with a specific IP address.

To run different pods in the Kubernetes cluster the best idea is to specify the number of copies of a pod using replicas instead of creating pods separately. Hence, the main intuition is to create a deployment and set up replicas to a distinct number of pods that should be run inside the cluster. Replicas are managed by the *Deployment* object in the Kubernetes that will be discussed in the next section. After creating pods using the deployment, all information related to a pod is passed to the API server. The following functions (see listing 4.9) in the form of an interface are primarily used to manage a pod within the cluster (kube batch n.d.).

```
type PodInterface interface {
        Create(*v1.Pod) (*v1.Pod, error)
        Update(*v1.Pod) (*v1.Pod, error)
        UpdateStatus(*v1.Pod) (*v1.Pod, error)
        Delete(name string, options *metav1.DeleteOptions)
        error
        DeleteCollection(options *metav1.DeleteOptions, listOptions metav1.ListOptions) error
        Get(name string, options metav1.GetOptions) (*v1.Pod, error)
        List(opts metav1.ListOptions) (*v1.PodList, error)
        Watch(opts metav1.ListOptions) (watch.Interface, error)
        Patch(name string, pt types.PatchType, data []byte, subresources ...string)
        (result *v1.Pod, err error)
        GetEphemeralContainers(podName string, options metav1.GetOptions)
        (*v1.EphemeralContainers, error)
        UpdateEphemeralContainers(podName string, ephemeralContainers *v1.EphemeralContainers)

        (*v1.EphemeralContainers, error)

        PodExpansion
}
```

Listing 6.9: The functions to manage pod within a cluster

In this thesis, we consider create and update functions for automation purposes. Other functions are enabled by the *kubectl* command to manage pods manually. The basic kubectl command to see the status of created pods is "kubectl get pods", which illustrates the status of running pods within the cluster. Indeed, the pods are shown in different statuses. The "Running" status confirms that the corresponding pod is successfully scheduled in a particular node. The "pending" shows that one or more containers of a pod are still waiting for execution. "Terminated", or "Failed " indicates that all containers of a pod are failed, or terminated.

Pods are accompanied by resource limits and requests as the fundamental indicators that allow the scheduler to find an appropriate node for a corresponding pod. In the next section, we will represent how resource limits and requests are specified in a deployment object. A limit is the maximum amount of a resource that can be consumed by a pod. Since all the containers share the total resources of a pod, the sum of the resource limits of all containers should not exceed the resource limit of that pod. However, resource limits are not considered in scheduling time. So, if a pod tries to consume more resources than its actual limit, then it could be evicted by the Kubelet and OSKernel.

6.3.3 Deployment in Kubernetes

Deployment in Kubernetes is a substantial pod management object, aiming to run applications and microservices. Scaling up-down, and rolling down are important use-cases of a deployment. It also specifies how many copies of a pod could be run within the Kubernetes cluster. A deployment employs the Yet Another Markup Language called YAML which is a human-readable language.

For example, if there is a web application that must be run on different machines, it is possible to specify the number of copies of that application in deployment, using replicas. Typically, a deployment provides automation in creating and updating pods without requiring manual operations. The deployment controller is responsible to manage, and control deployments. A deployment utilizes pod templates that compromise different specifications, for example, the number of replicas, types of deployment such as a service and replication controller. To create a deployment, kubectl is employed as a command-line interface that facilitates access to local, and remote clusters. It is also a gateway to interact with any Kubernetes cluster. Kubectl is mainly used to create, deploy, and manage any object within the Kubernetes cluster. The basic kubectl command syntax is as follows:

Kubectl [command] [TYPE] [NAME] [flags]

An example of deployment is shown as follows

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.8.1
        ports:
        - containerPort: 80
```

FIGURE 6.5: A sample deployment using YAML

As can be seen in Figure 6.5, the first element that should be specified is the apiVersion which indicates an application deployment running in API version v1 within the Kubernetes cluster. This is worth mentioning that the API version could be changed based on its corresponding version. The second part is the "kind" field which specifies the kind of deployment. In this example, the kind of deployment is "Deployment". In metadata, the name of the application that is *Nginx* is indicated. Nginx is a popular web serving open-source application which is mainly used for various operations such as load-balancing, and multi-media streaming. Within the "spec" field, different specifications are considered. Replicas are set up to 3 which means three copies of Nginx are eligible to run in different machines to serve user requirements. Labels in the template are a kind of identification for a given pod. The "spec" field in the template consists of the name of a container which also indicates that one docker Nginx container with version 1.7.9 is run inside the cluster. The containerPort "80" denotes a port to expose the pod to the outside world.

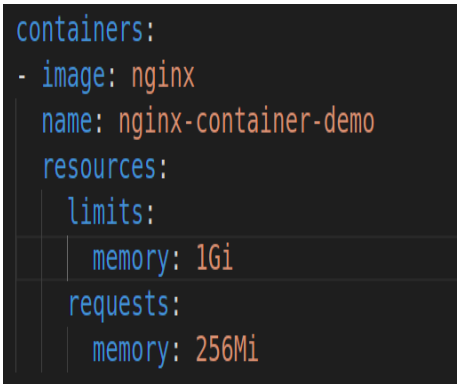


FIGURE 6.6: Deployment using resource requests, and limits

A deployment could have different configurations. In this thesis, we use deployments to create pods and replicas. Such a configuration is shown in Figure 4.8. In this example, one Nginx container is deployed, including resource requests, and limits.

A deployment is created using **"kubectl create -f PODNAME.yaml"**. Therefore, after creating it, *kubectl describe pod* command returns important information with regards to the created pod. To get the information regarding the deployment, the command **"kubectl describe deployment DEPLOYMENT"** is employed. This command provides necessary information about updating, scaling, and rolling up, and out operations. For example, by scaling replicas from 2 to 3, the updated information is represented using this command.

6.3.4 API Server

The API server in the master node is the core component that takes the control of worker nodes inside the cluster. Any alteration in pods and services is allowed by the API server. As it is shown in Figure 6.7, *etcd* is the central database within the master node that all information is stored in it, and accordingly, the API server regularly interacts with this database.

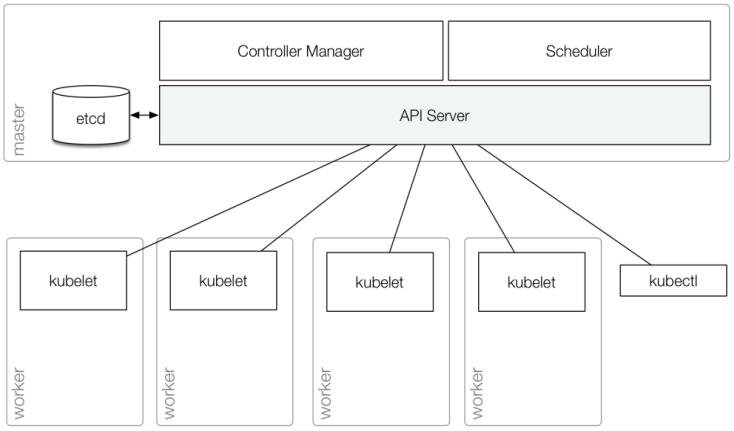


FIGURE 6.7: Kubernetes API server (Schimanski and Hausenblas 2018)

Once pods are created, the related information is passed to the API server, then the scheduler gets pods' specs from the API server and binds them to corresponding nodes. After scheduling pods, the scheduler, and kubelet within the worker node, update information with the API server. Different versions of API servers are considered by default that is: v1alpha1 which is typically disabled. The v2beta3 is activated for testing purposes, and the v1 version is supposed to be used for realizing the Software.

Almost all operations such as pod creation and scheduling are synchronized with API server version v1. In this thesis, we get the necessary information in terms of resources from the following functions, existed in **"/k8s.io/api/core/v1/"** (kube batch n.d.).

```
func (self *ResourceList) Cpu() *resource.Quantity {
        if val, ok := (*self)[ResourceCPU]; ok {
                return &val
        }
        return &resource.Quantity{Format: resource.DecimalSI}
}

func (self *ResourceList) Memory() *resource.Quantity {
        if val, ok := (*self)[ResourceMemory]; ok {
                return &val
```

```
        }
        return &resource.Quantity{Format: resource.BinarySI}
}

func (self *ResourceList) Pods() *resource.Quantity {
        if val, ok := (*self)[ResourcePods]; ok {
                return &val
        }
        return &resource.Quantity{}
}
```

LISTING 6.10: The functions for getting resource limits

The functions *CPU()*, and *memory()* returns resource limits for CPU, and memory as two primary resources in Kubernetes. Moreover, *pods()* function, gives all pods existed within the cluster.

In our proposed model, we get all necessary information from the API server which is shown as follows:

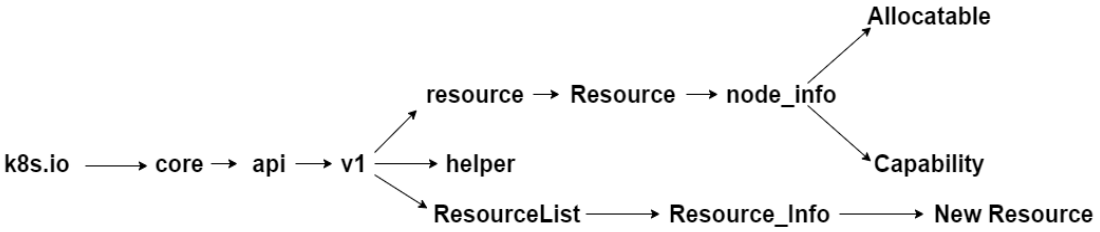


FIGURE 6.8: A tree that indicates how information is used to model our approach

Based on Figure 6.8 ,we use some parameters provided with the "node-info" object such as Allocatable, and capability. Allocatable refers to the available capacity of a resource. Also, a *Pod()* function is applied within the same object that returns all running pods within the corresponding node.

```
func (ni *NodeInfo) Pods() (pods []*v1.Pod) {
        for _, t := range ni.Tasks {
                pods = append(pods, t.Pod)
        }
        return
}
```

6.3.5 Scheduling in kubernetes:

The scheduling problem in Kubernetes is an optimization problem. It maintains a built-in scheduler called *kube-scheduler* which is an important part of the control plane. This component enables users to define their scheduling policies. Generally speaking, when a user creates a pod, resource requests and limits are also specified simultaneously. The scheduler considers different parameters such as predicates, priorities, and Quality of Service of pods. Predicates maintain functions as they get PodSpecs and node information like available resources and capacity of a node. Accordingly, they return a boolean value to indicate whether a pod can be fitted in a specific node. The priority of pods depends on the QoS classes of those pods. The importance of a pod is determined by the priority. Typically, pods with the lowest priority are the best candidate to be evicted (preemption), aiming to simplify scheduling the highest priority pods. Therefore, pods are categorized into three different levels: Guaranteed, burstable, and best-effort. The guaranteed class has the highest priority as all pods have the same resource limits and requests. In a case where the resource limit is set up above requested resources, it refers to the burstable class as a pod that can consume all resources up to the specified limit. Finally, in the best effort class, both resource requests and limits are not specified which makes a pod as a lowest priority one. The Kubernetes scheduler regularly tries to find a local optimum. So, it employs a multi-stage mechanism to schedule pods to appropriate nodes.

Typically, the scheduler in Kubernetes employs two steps to schedule pods in a set of feasible nodes that are filtering and scoring operations. The filtering operation attempts to find the most feasible nodes that can host a certain number of pods based on requirements and constraints. Technically, nodes are placed in a node list. If the node-list is empty, then the corresponding pod could be labeled as an unscheduled pod. Different parameters are contributed to filter nodes that are listed as follows:

- The scheduler regularly seeks for free ports with respect to each node. If any free port is found, then, the scheduler filters the corresponding node.
- Each node has its own specific hostname. So the scheduler filters the node with corresponding hostname.

- Every specific node in the node-list is filtered depending on available resources to host pod(s).
- The filtering is handled by matching a node selector, and pod label.
- The Volume check is consistently performed by the scheduler to determine if any request matches the volume in each node.
- The memory pressure is one of the most significant indicators which which specifies which node is suitable to host a set of pods. So, if there is any pressure on memory, then, a pod may not be scheduled in corresponding node.
- The same situation is applied to the disk as if there is any pressure on it, a pod could not be scheduled in any node with this condition.
- In some occasions, nodes are not ready to host a pod due to the networking issues. In that case those set of nodes will be filtered by the scheduler.

On the other hand, the scoring operation aims to rank nodes based on their feasibility to host a pod. Then, the most suitable node is selected to host that pod-based on its ranking score. The most relevant conditions that are considered in this thesis are listed as follows:

- The pods are deployed among hosts, taking into consideration pods fit in the same replica sets, and services.
- If there are many pods within a node with the highest resource utilization, those pods get the least ranking score.
- The pods with minimum resource utilization will get the highest ranking score.

6.3.5.1 Affinity and anti-affinity

The node-Selector introduces a best and simple approach to limit pods to nodes with specific labels called affinity/anti-affinity which develops a set of constraints as a user may represent. The affinity property involves two different types, “node affinity” and “inter-pod affinity/anti-affinity”. Node affinity is similar to node selector object, while inter-pod affinity/anti-affinity lets to limit which nodes are eligible to host-related pods.

Those pods could be scheduled according to labels that are already running within a node instead of labels on nodes. Node affinity is perceptually like the node selector that allows anyone to limit which pod is eligible to be scheduled in a particular node based on labels on the node. Furthermore, Node affinity is determined as a field called node affinity, placed within the PodSpec class.

6.3.5.2 Resource quota and namespaces

On top of the Kubernetes, and by deploying a plethora of services, managing simple tasks becomes more complicated. For example, teams unable to create services, or deployments with the same name. Accordingly, if there exist thousands of pods, it takes too much time to list all of them. Moreover, ff many teams are working in a shared cluster, it is necessary to assign them namespaces to ensure that each team benefits from a fair share of resources which is called quotas. Typically, a namespace refers to a virtual cluster within the Kubernetes infrastructure. It is possible to have multiple namespaces within the single Kubernetes cluster while they are isolated from each other. Namespaces also guarantee the isolation of each team's resources from the rest of the cluster. Kubernetes comes out with a default namespace. However, there are three different namespaces associated with Kubernetes that such as default, Kube-system, and Kube-public. Kube-public, though, has not many use cases in current architecture. So, it is usually a good idea to use the Kube system which is mainly applicable in Google Kubernetes Engine (GKE). On the other hand, the default namespaces are the only places that all pods and services are created. Namespaces are hidden from each other. So, service in one namespace may talk to services in other namespaces. Normally, when an application attempts to access a Kubernetes service, it is usually done by using a built-in DNS service discovery. Nonetheless, it is possible to create a service for the same name in multiple namespaces.

6.3.6 Resource allocation

Resource allocation could not happen directly in the Kubernetes cluster. It is performed by the Linux kernel OS and Linux Cgroups. In a simple definition, Cgroups is a control mechanism that limits resource consumption. As we discussed earlier in this chapter, resource limits are not considered in scheduling time. When a pod or a group of pods

are scheduled in a particular node, the kubelet passes pod information to Docker. Then, the Docker interacts with Cgroups to send out that information to Kernel OS through Cgroups. Finally, pods get resources based on specified limits. The allocation process is illustrated in Figure. 4.13.

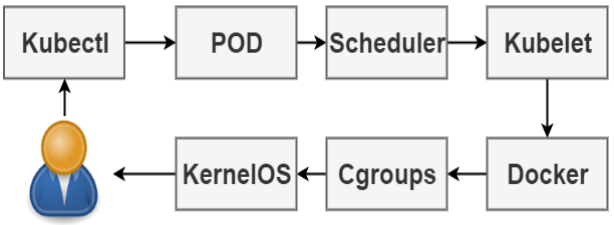


FIGURE 6.9: Resource allocation cycle in Kubernetes

Technically, if a pod tries to consume more resources than its specified limit, it could be evicted(killed) by the Kubelet. Typically, pod eviction is handled, regarding priorities based on the QoS of pods. If the total resource limits of all pods exceed the actual capacity of the corresponding node, the eviction process is commenced by the Kuebelet. The eviction has an adverse effect on QoS as a considerable number of pods may have to wait to be scheduled again in other available nodes. Subsequently, those pods with guaranteed QoS have the least chance for eviction. On the other hand, burstable pods have the highest probability to be evicted by the Kubelet.

One of the main objectives of this thesis is to manage resource limits in the scheduling time to avoid pod eviction. More specifically, we manage resource limits based on the notion of Dominant Resource Fairness(DRF). To prevent evictions, each pod is identified in a particular namespace. Then, pods with dominant resources in a specific resource type get the maximum resource limit subjected to the capacity constraints.

6.3.7 Kubernetes services

Kubernetes services facilitate communication between various components within and outside of the application. These services also allow users to connect applications with other applications. Consider an application that contains a set of pods, each of which runs different operations. For example, the first group for serving front-end workloads. The second for running back-end processes. And the third group, connecting to an external data source. In this case, the responsibility of a service is to enable connectivity among these pods. Moreover, services enable the front-end applications to become

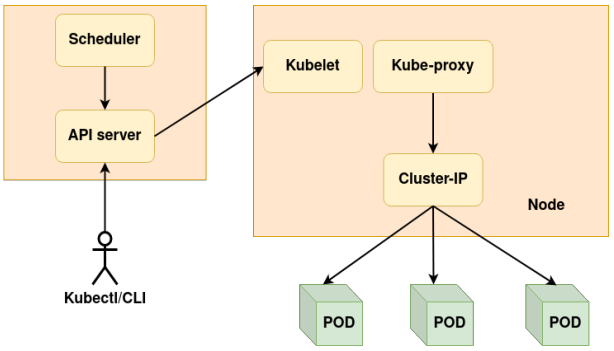


FIGURE 6.10: Kubernetes services abstraction

available to end-users. It helps communication between back-end and front-end pods. It also aims to establish connectivity to an external data source. In a nut shell, services enable loose coupling between micro-services within applications.

There are different kinds of available services in the Kubernetes. The first service is called *Node Port* in which the services make an internal pod that is accessible through a port on the corresponding node. The second type is the *Cluster IP* as the service creates a virtual IP inside the cluster to enable communication between different services such as the asset of front-end servers to a set of back-end servers. The third type is the *Load balancer*. The best example of the load balancer is distributing the load across different web servers. According to Figure 6.10, the Kube-proxy forwards the traffic to node-port service. Then the node port service is connected to pods through the ClusterIp service which is automatically created during node-port creation time. then, the Cluster IP service performs load balancing across all pods. The node-port and target-pod(related to each pod) are specified at service creation level.

In this thesis, we take the advantage of Kube-batch (kube batch n.d.) as a developed scheduler on top of the Kubernetes framework which is generally proposed to handle batch jobs and workloads. Figure 6.11, presents a high-level abstraction of the Kube-batch scheduler. Accordingly, we take into account the API, and Plugins object. To continue, we will show that how our proposed algorithms are integrated into the Kube-batch and Kubernetes infrastructure. Generally speaking, the best practice of implementing fairness algorithms in Kubernetes is implicitly shown in the Kube-batch project. As can be seen in the figure, different methods are integrated such as DRF, priority, Gang, and predicates. Each of these algorithms shows a specific functionality. For example, when a priority is not the case in the pod scheduling level, the DRF algorithm is performed.

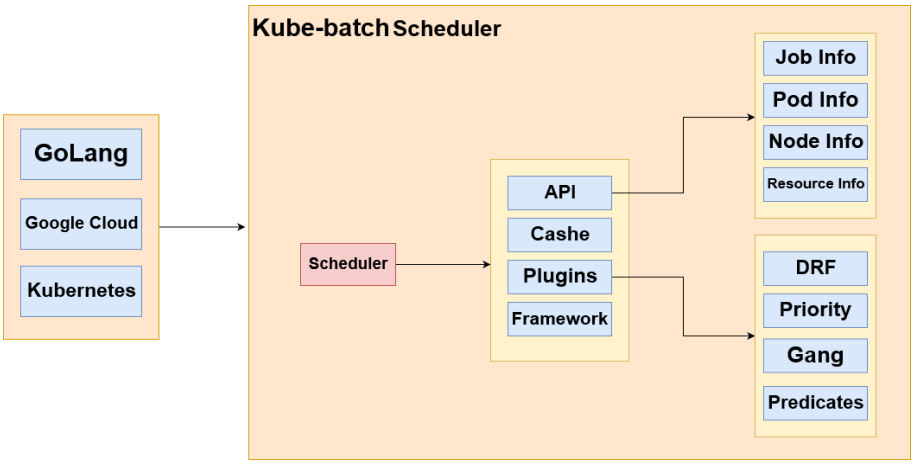


FIGURE 6.11: A high level abstraction of Kube-batch

6.3.7.1 minikube

Minikube is a tool that makes it easy to install and run Kubernetes locally on any operating system. Sometimes, it is necessary to set up all Kubernetes components individually in a local computer that could be a time-consuming process. Moreover, it is not desirable to consume system resources by creating multiple virtual machines. Therefore, to solve these issues, Minikube seems to be an appropriate solution. Minikube is responsible to bundle all components in the Kubernetes in a single ISO image. This image contains a preconfigured single-node Kubernetes environment. Hence, for setting up the Kubernetes, it is necessary to download an executable version of Minikube, and then install it. Once the image file is downloaded, the corresponding VM is created within the virtualization platform, including Oracle VirtualBox, or VMware (Jakóbczyk 2020). Minikube supports different Operating Systems such as Linux, Windows, and a variety of Hypervisors like Virtualbox, VMware Fusion, KVM, xhyve, and Hyper-V. Therefore, before starting the Minikue, at least one of the virtualization platforms should be installed in the system. In this thesis, the VMWare platform is used to run our proposed model. Once the minikube is configured, it needs a way to interact, and manage the Kubernetes cluster. This is done by installing another tool called Kubectl. The main purpose of using kubectl is to deploy and manage containerized applications.

6.4 Models for implementing proposed algorithms in Kubernetes

In this section, the architectural design of using proposed algorithms is represented in Kubernetes. We get all requirements from the high-level perspective of the system and define a framework to draw a straightforward approach for using algorithms in the Kubernetes. We try to implement a model as everyone can take it to implement our algorithms in their systems. Each algorithm is defined for a specific purpose that is represented in different types of diagrams.

According to the Activity diagram, shown in Figure 6.12, once a task is submitted to the system, the API server accepts user requests in the form of pods. Then the scheduler/allocator calculates dominant and non-dominant resources according to the provided information by the API server. The best practice here is how to choose different approaches to reach optimization goals. Accordingly, if the purpose is resource allocation, MLF-DRS, FFMRA , and H-FFMRA could be launched based on the requirements. MLF-DRS and FFMRA are great choices for resource allocation where the system is composed of a single resource pool, whereas H-FFMR is suitable for resource scheduling purposes.

The requirement diagram in Figure 6.13, represents a detailed description of applying fairness that could be generalized for every application area. On top of the designing process, the SLA requirement is the main functional requirement in which the utilization and fairness are its contaminants. In other words, the trade-off between fairness and utilization is a fundamental criterion in satisfying SLA for the system. To achieve this trade-off, it is required to apply optimization algorithms such as heuristics, Multi-objective, Langrangian multipliers, and swarm methods. In our proposed algorithms, we define two main optimization goals that are resource allocation and scheduling. To achieve these goals, dominant and non-dominant resources are calculated based on the node information in the Kubernetes that are derived by the API server. Also, the requested resources in the form of pods are submitted by users satisfied by real-time workloads. As it was stated before, the requested resources in the Kubernetes are composed of CPU and RAM as IaaS resources.

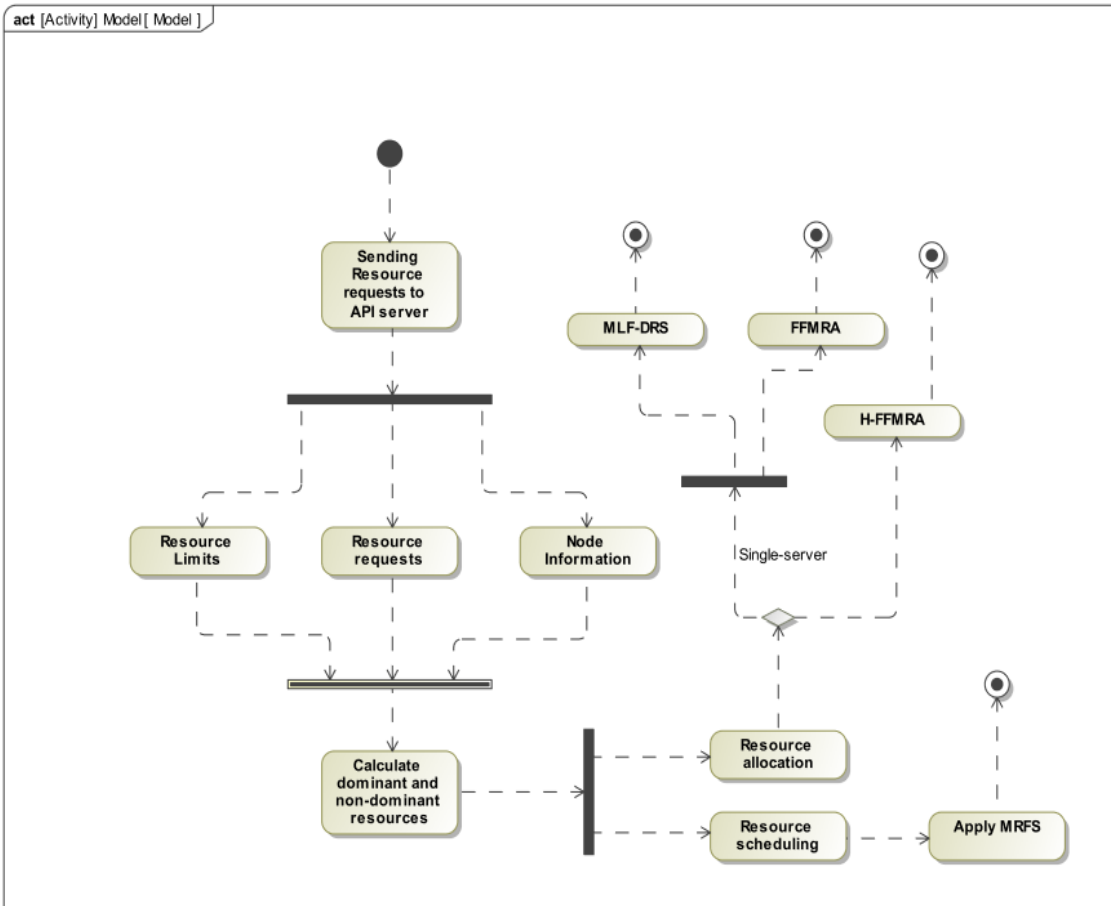


FIGURE 6.12: A block definition diagram that indicates how to select an appropriate policy based on the optimization goals

The sequence diagram in Figure 6.14. Illustrates the process of applying fairness in Kubernetes environments considering all proposed algorithms.

The diagram in Figure 6.15 clearly states that different approaches can be used based on the specified optimization purposes that are also shown in Figure 6.13. Hence, according to the optimization goals, a suitable approach can be used. This is worth mentioning that the purpose of this section of the thesis is to merely represent an approach to implement proposed algorithms in production environments.

Figure 4.20, illustrates a principle design of integrating fairness in the Kubernetes schedule. To have a further understanding of the model, we perform a quick review of the functionality of each component.

First of all, a user with a unique ID creates a pod using kubectl, including the requested CPU, and memory. The created pod with other information is passed to the API server which enables the Kubernetes API. The API consists of different components in which

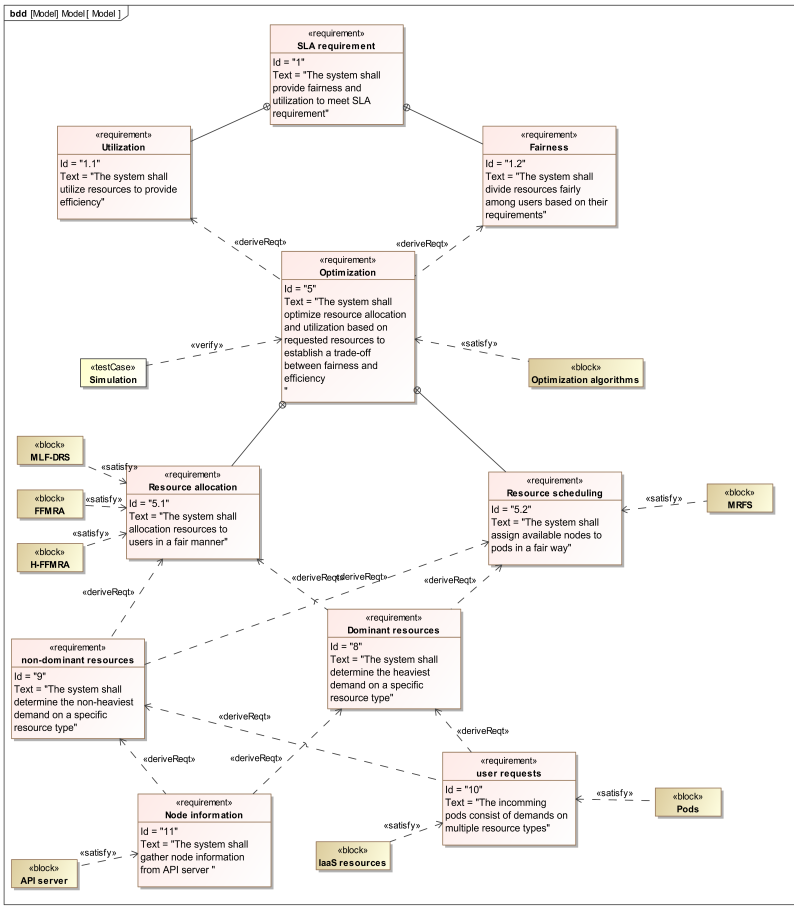


FIGURE 6.13: A requirement diagram that indicates how to integrate scheduling and allocation mechanisms based on the system requirements

the most important objects are nodes, pods, services, and types. When a node is created by a particular cloud provider that is referenced within the cloud provider object, the corresponding information is sent to the API server. Then other objects get the node information through the API server. According to the model, both scheduler, and policy broker get the required information from the API server. As we are trying to change and update resource limits, it is important to have node information and resource limits.

6.5 Summary

In this chapter, we extensively investigate the applicability of proposed algorithms in the CloudSim and Kubernetes frameworks. A model-based approach was taken into account, considering the whole architecture of both frameworks. The given models allow anyone to easily implement any fair resource allocation and scheduling mechanisms in either simulation and production environments.

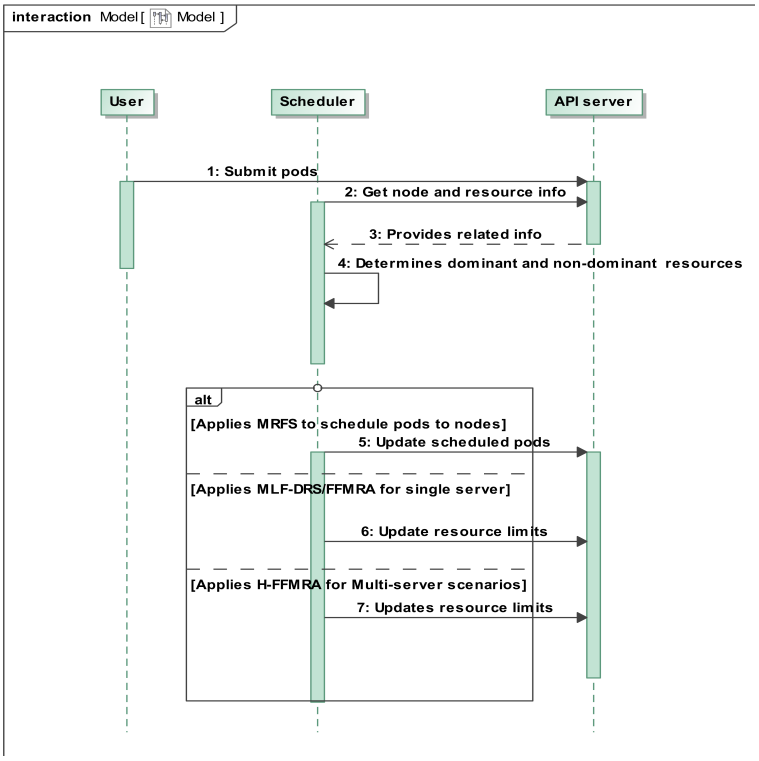


FIGURE 6.14: A sequence diagram that indicates how different algorithms could be applied in Kubenretes framework

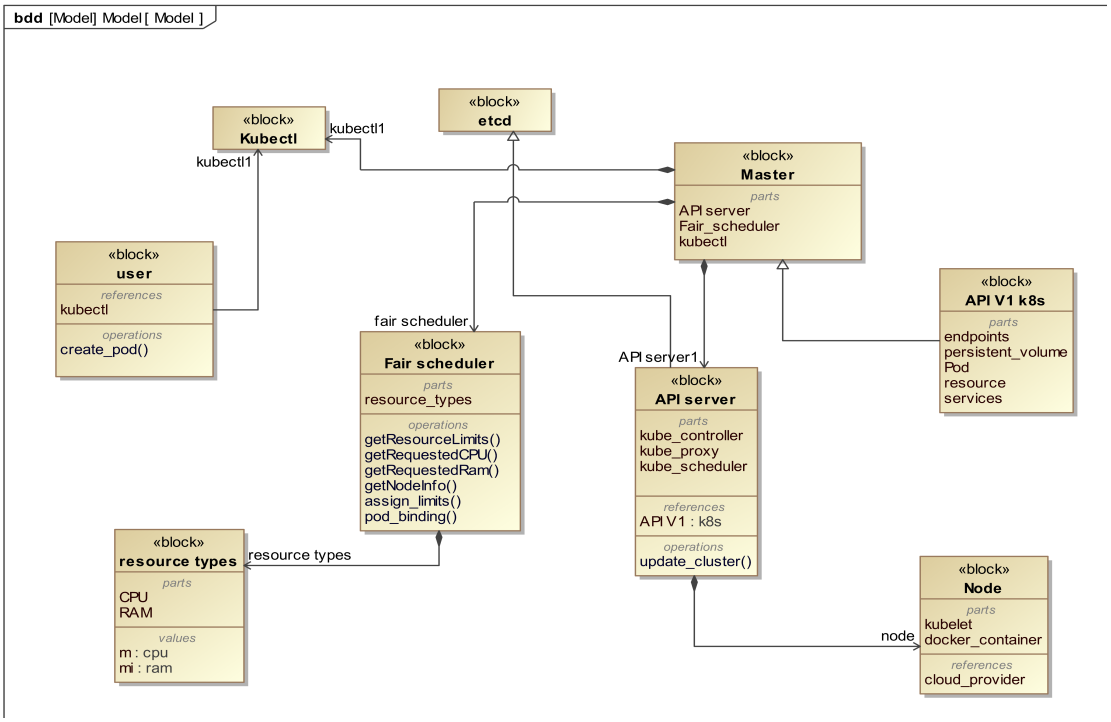


FIGURE 6.15: A block definition diagram, representing the integration of proposed algorithms in Kubenretes framework

Chapter 7

Evaluations and Results

7.1 Introduction

7.1.1 Workload

We apply trace-driven simulations to evaluate MLF-DRS, FFMRA, and H-FFMRA, using a stochastic data generator (Taddei 2015). Moreover, the MRFS is evaluated by the randomly selected workloads from the Google traces (Liu et al. 2016) by the sample length of 1% as the cluster is heavily loaded. The google cluster traces have been generally published in 2011, consisting of the traces of 12k servers that have been collected during 29 days. The workloads contain the statistics and configurations of heterogeneous resources like CPU, and Memory. Each job also contains one or more tasks, each of which is followed by a sort of demands. For simulations, the CloudSim is used as it was already discussed in chapter 4.

To achieve the random workloads, a class called stochastic has been created which carries out calculations to generate a set of various demands, considering different resource types such as CPU, disk, and Memory. The parameters considered within this class are minimum CPU, disk, and memory. Standard deviation and vertical stretch are also employed as mathematical indicators to specify requested resources. Besides these indicators, other parameters are also taken into accounts, such as resource consumption on average and the standard deviation of different resource types. To evaluate proposed

algorithms, we normally consider four important parameters: fairness, resource allocation and utilization, and scheduling time. Jain's index is employed to measure fairness. Furthermore the proposed β fairness index that was already discussed in 2.2.5, 4.24, and 4.25 respectively. Jain's index is considered to evaluate MLF-DRF as it is well-adopted concerning each specific resource type, While, β is more applicable to evaluate FFMRA.

The processing workloads in the CloudSim are mainly time-dependent. Based on the number of workloads within a server, the time could be varied. The small number of tasks may result in increasing the overall time. During the experiments, there are cases where the workloads are not completed at the same time, as there are idle times associated with a server. As a result, the efficiency may not be well-respected compared to real-world scenarios.

7.1.1.1 Workload Generation

Statistical models are used to generate random workloads. Several indicators such as degree, Gaussian, median, deviation, standard deviation radiance, and stretch are taken into consideration to determine resource requests based on CPU(Mips), RAM, and storage disk. To calculate demand profiles for these resources, it is necessary to specify all mentioned indicators. Radiance and degree are tightly coupled indicators. Workload variation is depended on these variables. Hence, a degree can be determined as follows:

$$\theta = \frac{s}{r} \tag{7.1}$$

Which θ refers to a degree of an angle (Radian) in a spatial environment; s denotes the distance between two points, and r indicates the value of the radius.

Therefore, based on 7.1, the degree is calculated as follows:

$$degree = \theta.\frac{180^{\circ}}{\pi} \tag{7.2}$$

After calculating the degree in 7.2, the following formulation determines deviation.

$$deviation = sin(\theta.degree) \tag{7.3}$$

The next step is to determine the standard deviation s according to 7.3 based on the real sample of workloads.

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \tilde{x})} \tag{7.4}$$

Hence, based on 7.4, N refers to the total number of observed workloads, $\tilde{x}$ is the mean value, and x_i is any workload in a set X.

The Gaussian is another indicator that is used to generate workloads. To reach the Gaussian value, the following formulation is used.

$$P(x) = \frac{1}{\delta\sqrt{2\pi}}.e^{-\frac{1}{2}(\frac{x-\mu}{\delta})^2} \tag{7.5}$$

Each resource type in the workload generation scheme has to fluctuate at certain point. To achieve this, it is required to use the value of median which is generally determined as follows:

$$median(a) = \frac{a\lfloor \bowtie a + 1 \div 2 \rfloor + a\lceil (\bowtie a + 1) \div 2 \rceil}{2} \tag{7.6}$$

According to 7.6, a is an ordered set of workloads, and $\bowtie$ is the length of any workload within set a. In the final step, we need to specify the value for the stretch. Generally speaking, the stretch refers to a function that determines the difference between a dominant, and non-dominant resource. Setting up the variable say i in a stretch function, increases the difference among dominant, and non-dominant resources. The stretch in a coordination vector is shown in Figure 7.1.

Consequently, following the above-specified indicators, it is time to generate workloads for specified resources.

Mathematically, the CPU is calculated by applying the normal distribution formula based on 7.5 which is a Gaussian value based on median, stretch, and standard deviation. Hence, we have:

$$f(x) = Mips \tag{7.7}$$

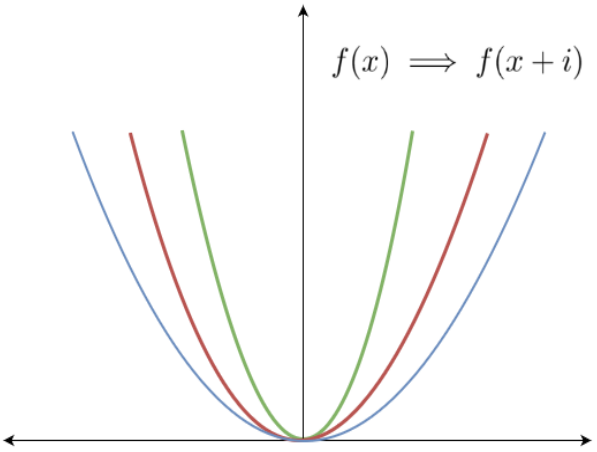


FIGURE 7.1: Vertical stretch

Assuming the dependency among other types of resources with CPU, the workload generation for RAM, and disk storage is shown in listing 5.1.

```
RAM = 260 + 100 * deviation*stretch
Disk storage = Mips / 2.0
```

LISTING 7.1: Workload generation rule for RAM

7.1.2 Vm configuration

The basic VM configuration for all rounds of experiments is depicted in Listing 5.2. Correspondingly, the VM type is Xen in which the default capacity of RAM is set to 512 MB. This is important to confirm that regardless of how much the requested resource is, the VM size can be scaled up or down depending on the actual resource demands by users.

```
public class VmConfiguration {

        private String vmm = "Xen";
        private int mips = 300;
        private int ram = 512;
        private int bw = 1000;
        private int peCnt = 1;
        private long size = 10000;
}
```

LISTING 7.2: Vm configuration for all experiments

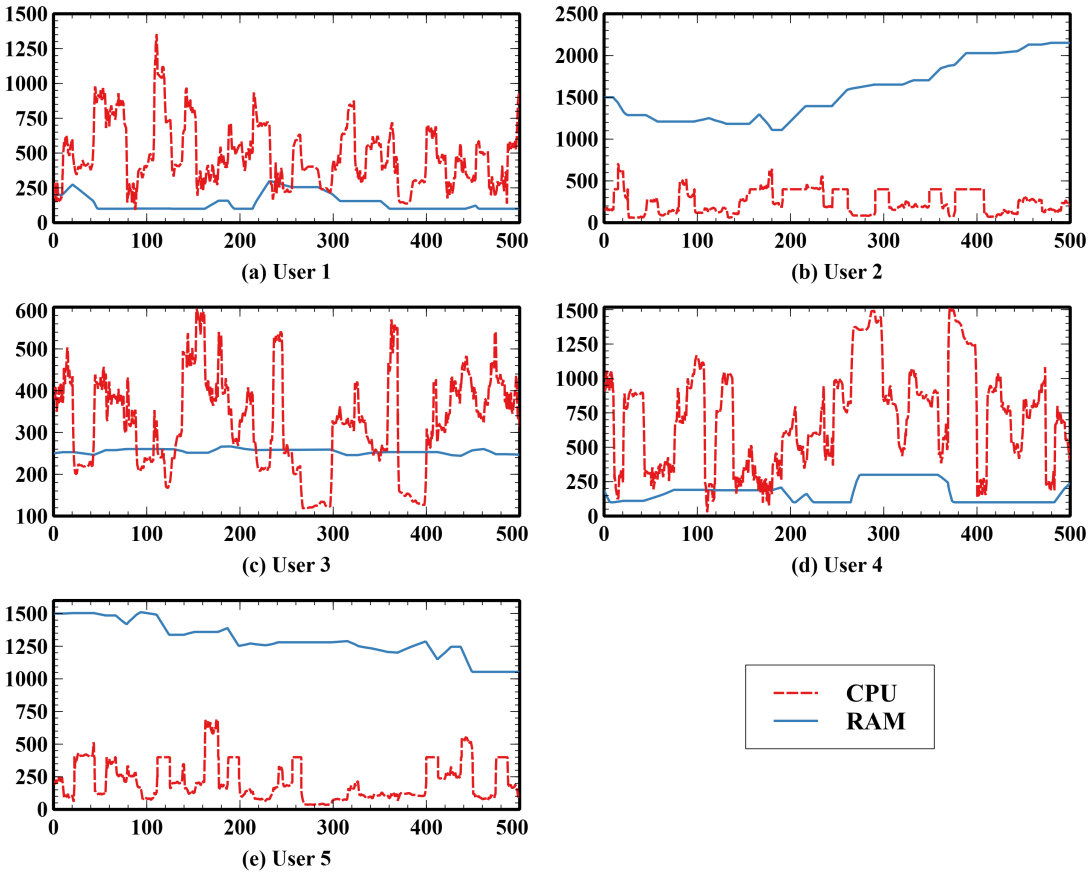


FIGURE 7.2: Sample workload generation based on requested resources, taking into account CPU, and memory

Figure 7.2 represents sample generated workloads, using the above mathematical notations. This is actually counted as requested resources for five users (CPU and memory in this time-series experiment). In this figure, the variation in CPU-based requests is strictly much compared to requests on memory. Accordingly, users tend to submit a various range of requests on CPU. In continue, we will show that how this variation in workloads could affect decision making in resource allocation in different mechanisms.

7.2 Evaluations: MLF-DRS

In this section, the performance of MLF-DRS is evaluated, considering four parameters: resource allocation, utilization, fairness, and scheduling time. Taking into account that MLF-DRS tries to allocate resources based on dominant resources on each specific resource type, CPU and RAM are considered. Hence, the resource demand profile for each user is expressed as $R = (r_1^1, r_1^2)$. To do the experiments, we consider five users, submitting their tasks with various demand profiles (Saying that all demands are positive as $x_{ik} \geq 0, \forall n \in U, k \in R$). The experiments are configured in 1000 iterations, using a server that has been configured based on Table 7.1. Similar to the notion under DRF policy, it is assumed that all resources are concentrated in a shared resource pool rather than a bunch of servers.

TABLE 7.1: Server configuration

Resources	CPU	RAM
Capacity	2000	4098

7.2.1 Resource Allocation

Figure 7.3, presents the resource allocation in MLF-DRS, considering two resource types: CPU and RAM. The decision-making process that was already discussed in 4.6 is applied in all rounds of experiments. Users may represent different behaviors in the job submission time, as various sets of submitted tasks could be dominant on a particular resource type. Hence, the allocated resources may be varied based on ($x_i^k \leq f^k$, or $x_i^k \leq f_s$). We will show that this allocation has a direct effect on fairness in sharing multiple resources among users.

In Figure 7.3, the submitted tasks by users are reported. According to the figure, the submitted tasks by users 1, 3, and 5 are always CPU-intensive(dominated on CPU). Based on Figure 7.4(a), in some iterations, the allocation for at least two users is the same. This is due to that the requested resources by users are less than the *fair-share*. Nonetheless, in a certain period of time, the allocation is not the same for tasks with dominant CPU as$(u_i(\Pi_i^{'}) > u_i(\Pi_i))$. Consequently, a user's utility could be lower or greater than other users' utilities with a dominant CPU.

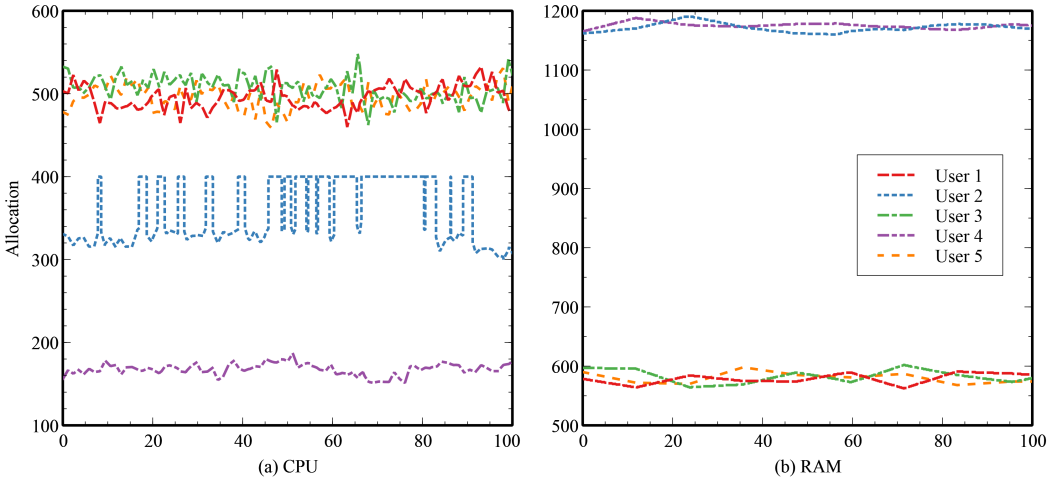


FIGURE 7.3: Resource allocation in MLF-DRS

As can be seen in Figure 7.3(b), all submitted tasks by users 2 and 4 are RAM intensive. This may result in maximizing allocations for these users. Also, approximately in 50% of iterations, the allocation is the same for both of them. Figure 7.4 compares RAM allocation in DRF and MLF-DRS. A comparison between Figures 7.5(a)(c), and 7.5(b)(d)(e) reveals that under MLF-DRS users with RAM intensive tasks, get maximum share compare to DRF. It can be seen that MLF-DRS considers minimum resources to users 1 and 3 in contrast to DRF as both of which has a non-dominant task on RAM. Hence, users 2, 4, and 5 with dominant RAM benefit maximum share under MLF-DRS compared to DRF.

7.2.2 Resource utilization

In Figure 7.6, we present the total utilization of resources that is achieved on average over 10 times experiments in 1000 iterations with 1000 users. The figure states that DRF and FFMRA show a better utilization of CPU as in some range of iterations, the utilization for both of them are approximately the same. However, in some intervals, the utilization of MLF-DRS outperforms DRF. As expected, and according to Figure 7.6(b), MLF-DRS establishes a significant improvement in RAM utilization compared to DRF.

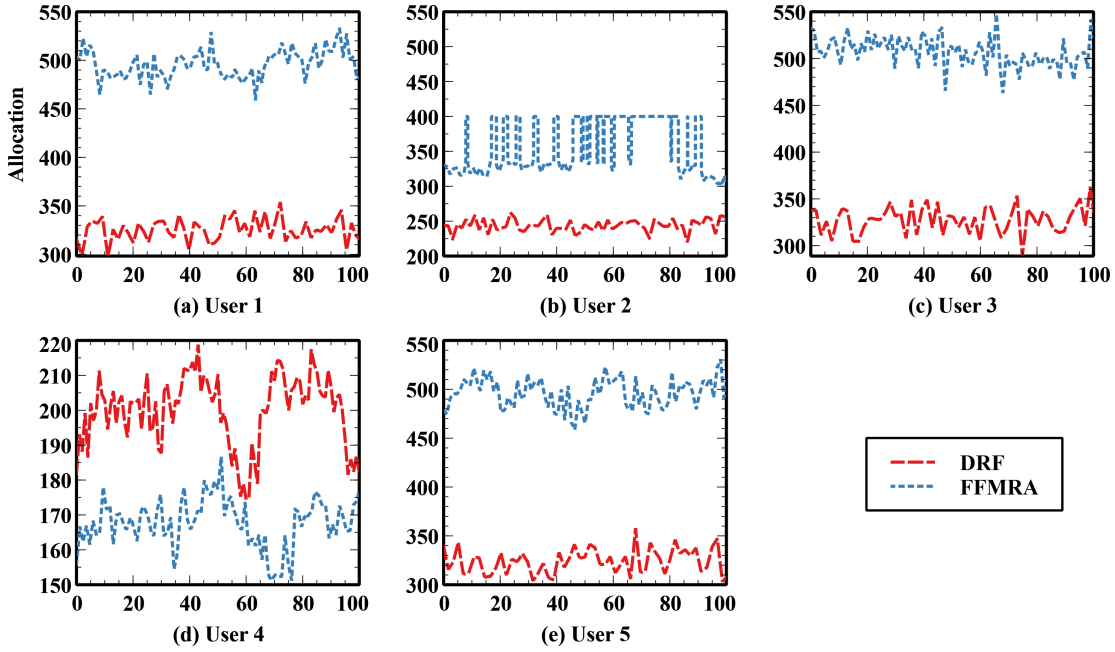


FIGURE 7.4: CPU allocation in MLF-DRS

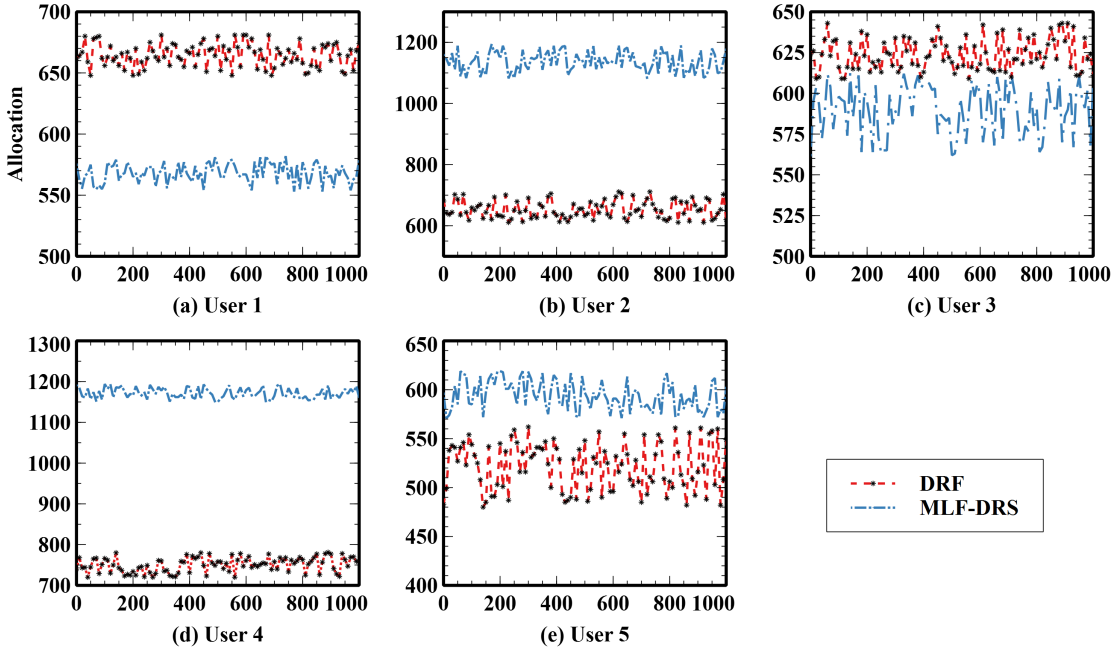


FIGURE 7.5: RAM allocation in DRF and MLF-DRS

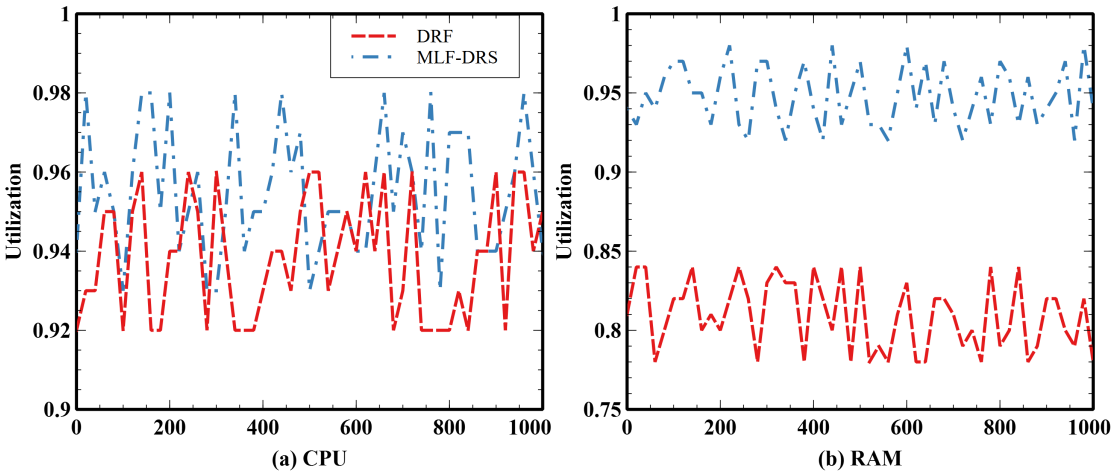


FIGURE 7.6: The comparison of resource utilization in DRF and MLF-DRS

7.2.3 Fairness

As we mentioned in 2.2.5, the *Jain's* fairness index is one of the most common approaches to evaluate fairness in allocating resources. Although, the bare application of Jain's fairness is used for the single resource scenarios, it is also possible to apply it to dominant resources in Cloud computing. Therefore, to evaluate the fair sharing in MLF-DRS, we employ Jain's index, conducting 10 times experiments in 1000 iterations with 1000 users.

The fairness index for CPU and RAM allocation is illustrated in Figure 7.7. Based on Figure 7.7(a), MLF-DRS has an advantage over DRF in sharing resources fairly among users. This improvement continues by iteration 800, however, from iterations 800 to 900, DRF shows better fair allocation compared to MLF-DRS. This is due to that at this interval, the value of demands with a dominant resource on CPU is greater than the fair share. Consequently, and corresponding to the formulation in 4.6, the allocation for users with the dominant resource is not the same.

fairshare.

Figure 7.7(b), presents the fair share of RAM among all users. As can be seen in the figure, in some iterations the allocation is approximately the same in DRF and MLF-DRS. However, in other intervals, MLF-DRS behaves considerably fairer.

The resource pool capacity and the number of users have a direct impact on fairness in MLF-DRS. To realize this, we conduct six experiments with an identical number of users and diverse resource capacities. The $jain's index$ is calculated in average for each

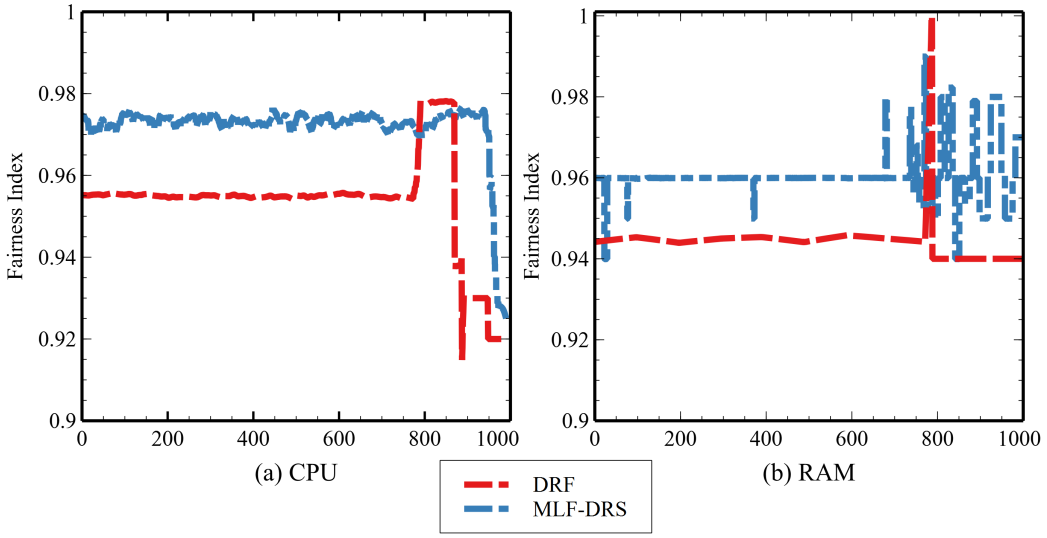


FIGURE 7.7: Jain's fairness index to evaluate how resources are fairly shared among users in Users in DRF and MLF-DRS

experiment in 1000 iterations. The requested resources by users are normalized by $1/r_i^k$ as many users can submit their tasks to the system. As can be seen in Figure 7.8, by increasing the capacity of servers, Jain's index is increased accordingly. This is due to that by increasing the capacity of the resource pool and taking into account that MLF-DRS allocates resources based on the fair share, a majority of submissions are less than the fair share. Consequently, MLF-DRS tries to allocate the same amount of share for those tasks.

7.2.4 Scheduling Time

In this section, the finishing time of submitted tasks corresponding to each user is determined under DRF and MLF-DRS mechanisms. For the sake of simplicity, an experiment is conducted with 20 users in 1000 time series. There is no significant improvement on completion time, nonetheless, for some tasks, arrived after several tasks with different dominant resource types, those tasks are required to wait a long time in the queue to get scheduled. While DRF schedules tasks with the lowest dominant resource in each iteration, FFMRA tries to schedule the first coincidence of any submission with the dominant resource type after many occurrences of tasks with the same type of dominant resource.

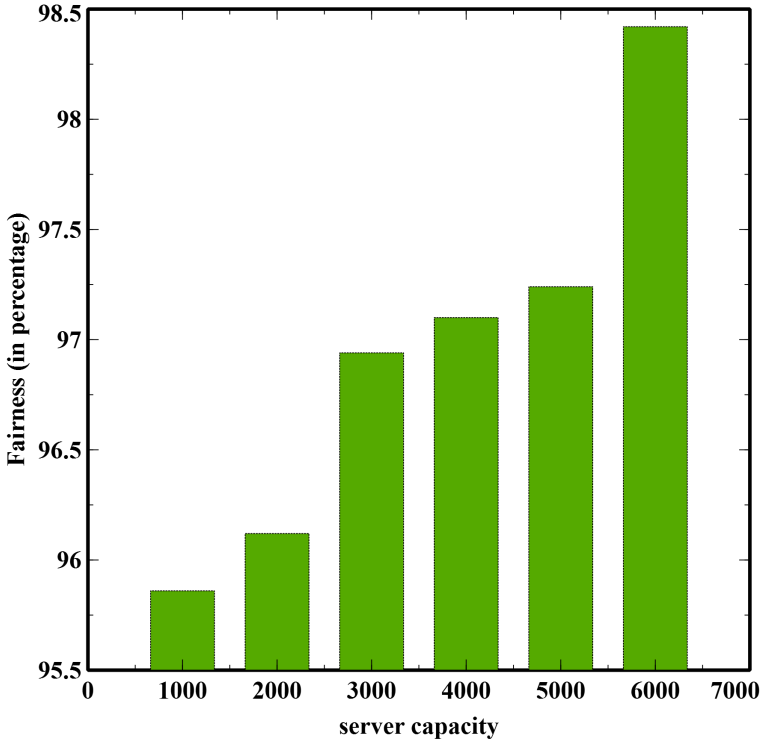


FIGURE 7.8: The effects of server capacity on equal allocation of resources to tasks with dominant resource on a particular resource type under MLF-DRS policy

TABLE 7.2: The comparison of scheduling time in DRF and FFMRA

FFMRA									
user_8	**user_2**	**user_14**	**user_15**	**user_12**	**user_11**	**user_3**	**user_6**	**user_18**	**user_0**
499.81	499.97	500.02	500.15	500.25	501.15	502.55	502.85	507.85	507.92
user_19	**user_9**	**user_1**	**user_7**	**user_4**	**user_16**	**user_13**	**user_5**	**user_10**	**user_17**
509.78	512.53	518.93	527.01	579.62	573.92	589.55	610.77	630.03	647.55
DRF									
user_2	**user_11**	**user_14**	**user_5**	**user_8**	**user_17**	**user_3**	**user_6**	**user_15**	**user_9**
500.02	500.02	500.02	500.03	500.03	500.03	503.26	503.46	504.35	507.91
user_12	**user_0**	**user_18**	**user_13**	**user_7**	**user_1**	**user_4**	**user_10**	**user_16**	**user_19**
508.25	508.54	517.29	572.68	579.39	585.69	587.67	610.84	630.34	641.17

Overall, the scheduling time under MLF-DRS, and DRF is shown in Table 7.2. Accordingly, the scheduling time for users is varied depending on DRF and MLF-DRS policies. For instance, MLF-DRS first schedules $user_8$' tasks, while in DRF it is scheduled after four users. This is because of the different approaches that both mechanisms employ to schedule tasks. As another example, $user_{15}$ is scheduled earlier in MLF-DRS than DRF, as $User_5$ has the lowest dominant resource than $User_{15}$. However, since $User_{15}$ is the first occurrence of dominant resource on disk, MLF-DRS tries to schedule it much earlier than DRF. Therefore, since MLF-DRS employs separate queues for tasks with a dominant resource on a specific resource type, users benefit from an equal opportunity to get scheduled based on their arrival time. To conclude, it is worth mentioning that

even though, fairness and efficiency in resource allocation play a significant role, however, fair queuing in cloud computing with regards to dominant resources needs to be more investigated.

7.2.4.1 A scenario with five users

In this section, an experiment is conducted, taking into account five users in 1000 iterations. The requests are randomly generated each as a single task. In other words, the submitted tasks are arbitrarily generated in which the demands are different in terms of dominant resources in each iteration. Figure 7.9 represents the fraction of submitted jobs over the actual capacity of each resource type (Here are CPU and RAM). Accordingly, the figure states the dominant resource of each task in different time series.

According to the figure, users have different preferences in each iteration. Consequently, the scheduling time for each user could be different as MLF-DRS schedules a task in a short time with minimum occurrences of the same type of a dominant resource.

Figure 7.10 demonstrates the response time for submitted tasks by five users over 1000 iterations. As an example, we consider the first 100 iterations to figure out how the number of dominant resources may affect the response time. As we already discussed, MLF-DRS schedules tasks in different queues based on the type of dominant resource. According to Figure 7.9, users 1, 2, and 3 expose interest in submitting RAM-dominated tasks while users 4 and 5 submit CPU-dominated demands. Consequently, user 3 as the first user with an early arrival time, is scheduled first. Then, since user 4 is the first one with CPU-dominated tasks in front of the queue for CPU-intensive tasks, it is scheduled earlier than other tasks. This is a note that, although users 1 and 2 arrive earlier than users 4 and 5, their tasks are scheduled in the next stage.

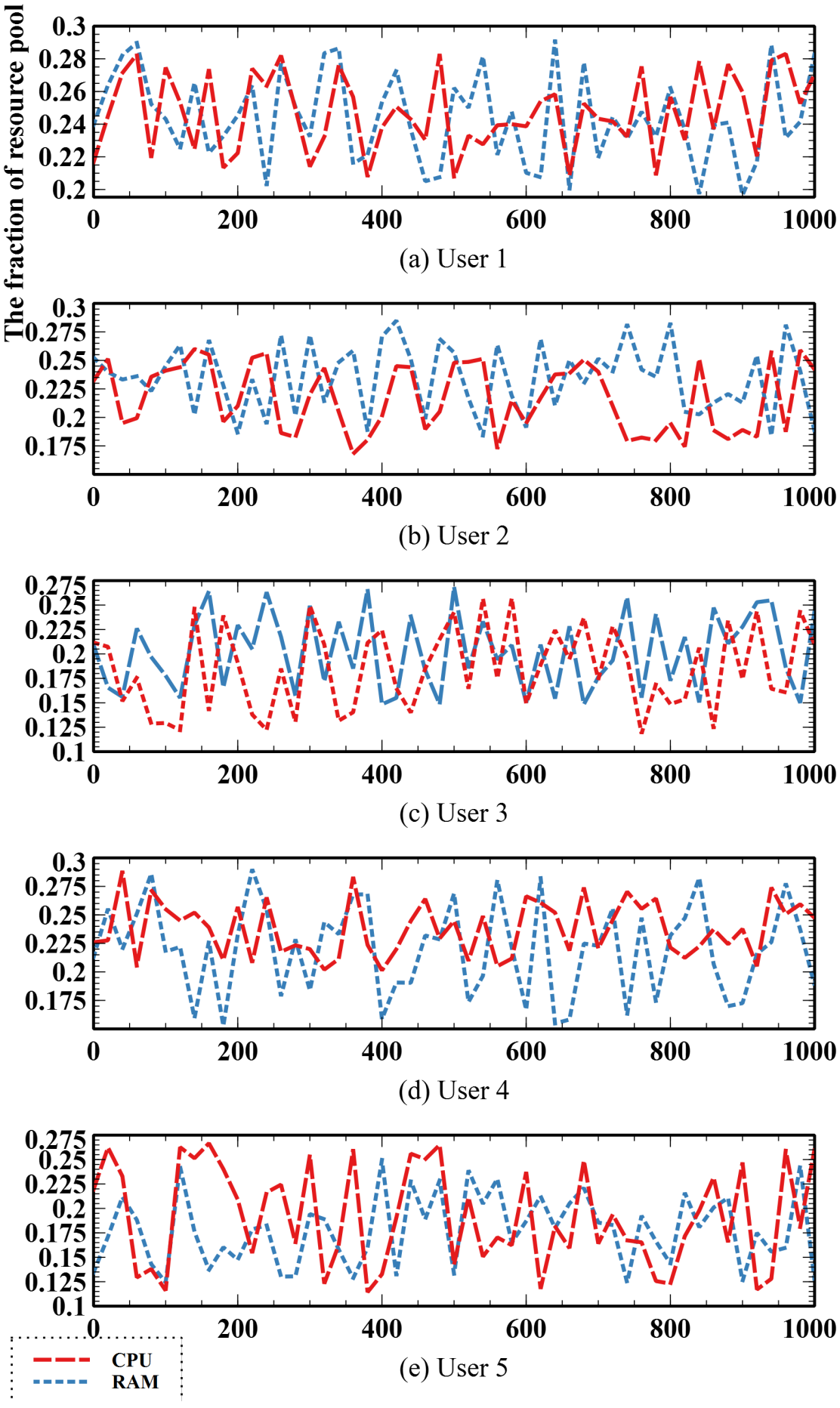


FIGURE 7.9: Dominant CPU and RAM for each task in 1000 iterations. The higher value means the corresponding task is dominated on CPU or RAM

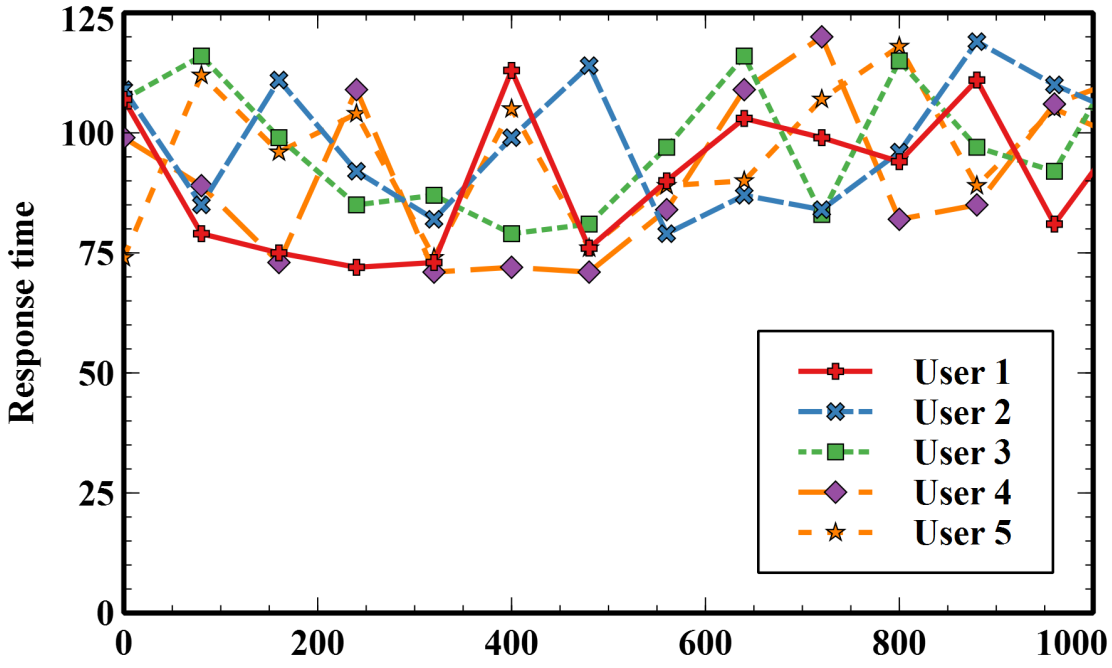


FIGURE 7.10: The response time with respect to the submitted tasks either dominant on CPU and RAM.

7.3 Evaluations: FFMRA

In this section, we go through the evaluations of FFMRA, taking into account different metrics: fairness, resource allocation, and resource utilization. We compare FFMRA with DRF, considering three different resource types: CPU, RAM, and StorageDisk as the demand profile is $R = (r_1^k, r_2^k, r_3^k)$ in which $K = (CPU, RAM, disk)$. We also conduct experiments considering a certain number of users and a server with a distinct configuration which is shown in Table 7.1. Moreover, the submitted tasks from the users are fixed and indefinitely divisible.

To get the expected results under the notion of FFMRA, we set the standard deviation and vertical stretch to a high value to maintain the maximum difference between dominant and non-dominant resources. This configuration can be applied when there are a small number of users, as in our experiments it is not possible to compare the allocation for a large number of users. However, in a real scenario with many users, it is not necessary to pre-define mentioned configuration.

The frequency of tasks with dominant resources is illustrated in Figure 7.11. According

TABLE 7.3: Server configuration for five users

	CPU	RAM	Storage
Capacity	20000	10000	12000

to Figure 7.11(a), it is obvious that in almost 50% of iterations, 75% of demands are dominant on CPU. Due to the server configuration and generated workloads, the frequency in RAM does not go above 3 as in most of the cases it stays at 2 which is clearly shown in Figure 7.11(b). On the other hand, and as can be seen in Figure 7.11(c), although the frequency in the dominant disk is more than RAM, in most intervals the frequency reaches 0. So, we conclude that CPU and RAM are the most frequent in terms of dominant resources.

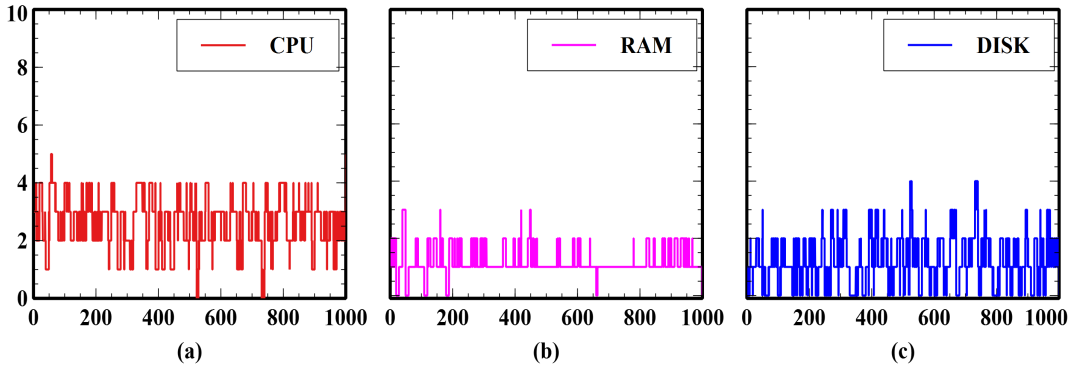


FIGURE 7.11: The frequency of dominant resources

7.3.1 Resource allocation

To evaluate the performance of FFMRA regarding resource allocation, we conduct an experiment with five users over 1000 iterations based on milliseconds. As the dominant resources of users' demands are changed during the time, the allocated resources are also changed accordingly. The experiments show that the submitted tasks by users 2, 4, and 5 are consistent and almost always CPU-intensive. However, in a different range of iterations, the tasks for users 1 and 3 become dominant in CPU while others are non-dominant. It results in maximizing the allocation for users 1 and 3 and minimizing for others. The requested resources by users are shown in Figure 5.8. The values in y axis are normalized by $(requestedresources/resourcecapacity)$. Accordingly, a higher value means the dominance of a specific resource type over other resources concerning the capacity of the resource pool. According to Figure 7.12(e) and (b), in the first 350 iterations users 4, and 5 have a dominant resource on CPU. As a result, they get

approximately the same unit of CPU as their requested resources are less than the *fairshare*. Figure 7.14 clearly states that how FFMRA allocates resources fairly among users with dominant resources. Particularly, according to Figures 7.14 (d)(e), as the demands for users, 4 and 5 are CPU-intensive, FFMRA allocates an equal share to both users under the circumstances mentioned in Algorithm 3. In other iterations, since there is no consistent type of requested resources either dominant or non-dominant, the allocation could not be equal. The equal share among these users is also obvious in iteration 850. However, over iteration 450, although both users have a dominant submission on CPU, as the demand by user 5 is greater than *fairshare*, hence, the allocation is also higher than user 4. Furthermore, as can be seen in Figures 7.12(b) and 7.12(b), by growing the time, and after iteration 400, the majority of submissions by user 2 are CPU-intensive that lead to maximizing the allocation for this user. The figure also confirms that the overall allocation under FFMRA policy is significantly better than DRF concerning intensive tasks. The behavior of some users in submitting tasks is interesting as based on Figure 7.12(a),(c), and (d), there is a fluctuation in submitting dominant and non-dominant tasks. The case for user 1 is more obvious, wherein a specific range of iterations, the user tends to submit different types of tasks. On the other hand, FFMRA allocates the equal share of CPU for non-dominants as in most of the iterations the requested resources for non-dominant demands are usually less than the *fairshare*. For instance, as can be seen in Figures 7.14(a)(b), users 1 and 2 get exactly an equal share of CPU between iterations 100, and 350.

In terms of RAM allocation, and as can be seen in Figure 7.12(c), user 3 is the only one who tends to submit RAM-intensive tasks. Also, based on Figure 7.15(c), this user gets the maximum share compared to other users. In more detail, over the first 70 iterations users 3 get the maximum share of RAM, however, between iterations 70, and 100, user 1 starts to submit RAM-intensive demands. Consequently, and according to Figure 7.14(a)(c), FFMRA tries to consider equal share to both users. As the frequency of demands with dominant RAM is trivial, so, in most intervals, the coincident of two or more users with RAM-intensive tasks over the same iteration is correspondingly rare. Consequently, if there is only one demand with a dominant resource on RAM, FFMRA allocates the maximum share which is considerably higher than DRF. This is a note that, in specific cases such as an example in Table 7.4, when the proportion of resource pool to dominants is higher than non-dominants, FFMRA gives a maximum allocation

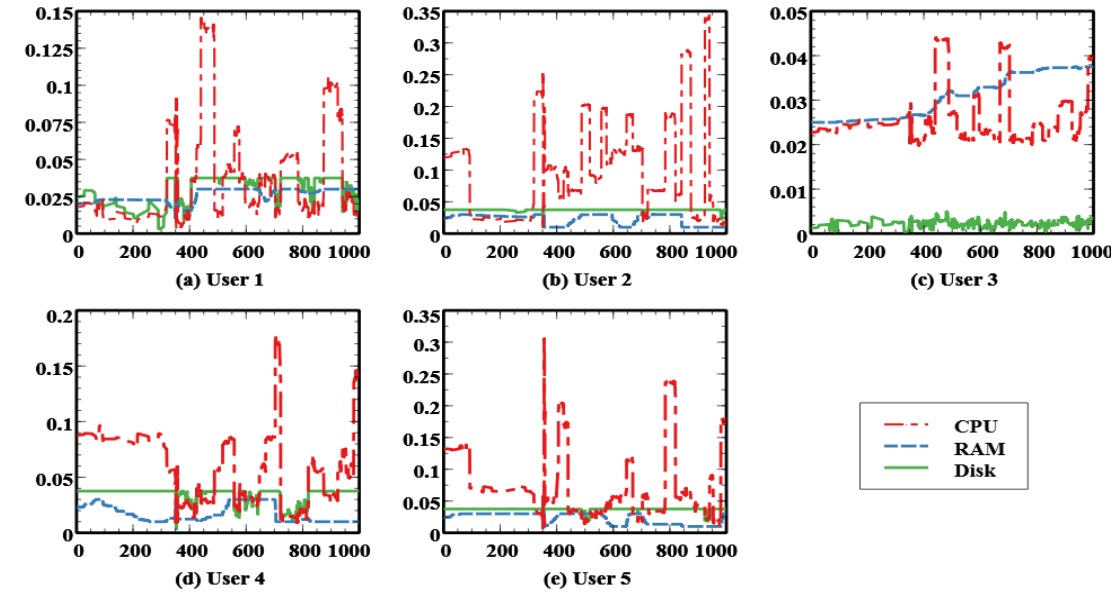


FIGURE 7.12: The fraction of requested resources by the users over the capacity of server

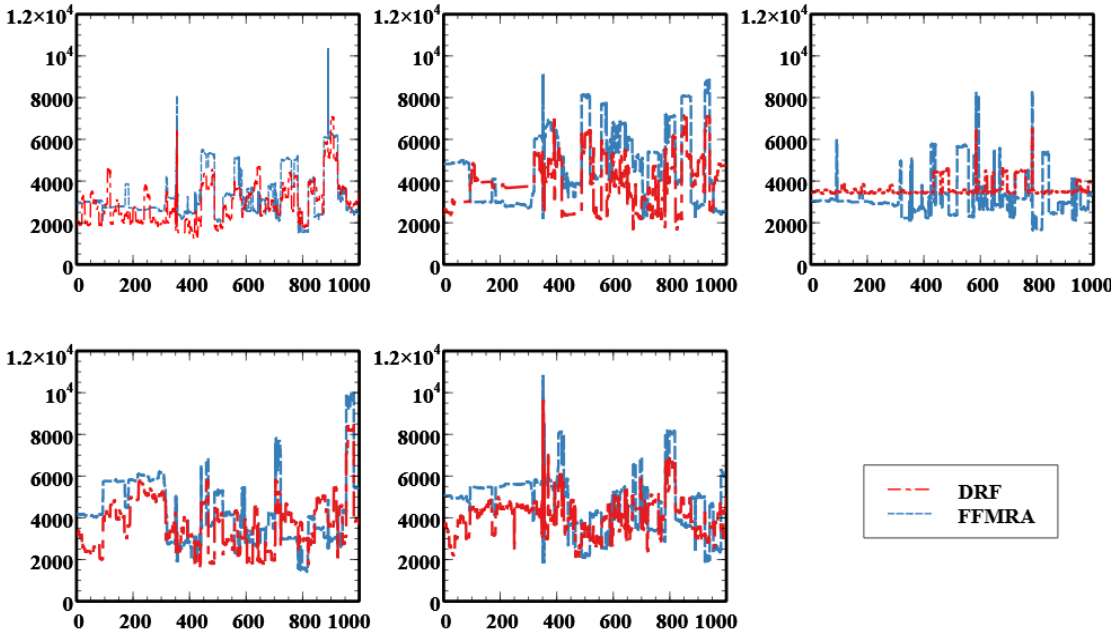


FIGURE 7.13: Allocated CPU for all users in DRF and FFMRA

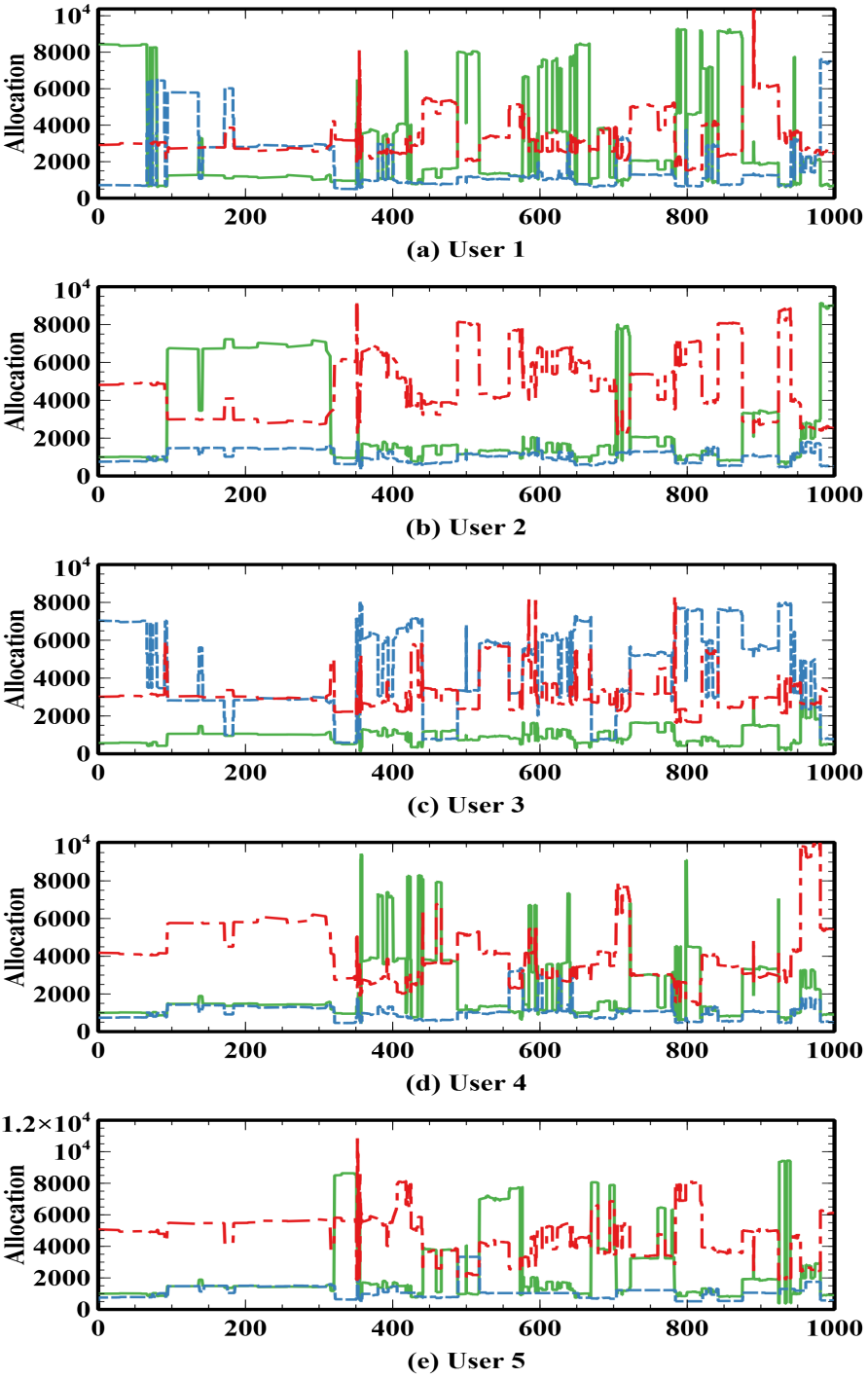


FIGURE 7.14: The comparison of allocated resources under FFMRA policy

to users with dominant resources.

On the contrary, Figures 7.12(b)(d)(e) report the majority of non-intensive submissions with respect to RAM in almost all intervals. Exclusively, over iteration 500, user 5 requires to schedule RAM-intensive tasks. At the same time, user 3's submission is also dominant on RAM. As a result, FFMRA considers equal share for both users. Nonetheless, as user 3's requested resource is quite more than *fairshare*, therefore, the allocation for user 3 is slightly higher than user 5.

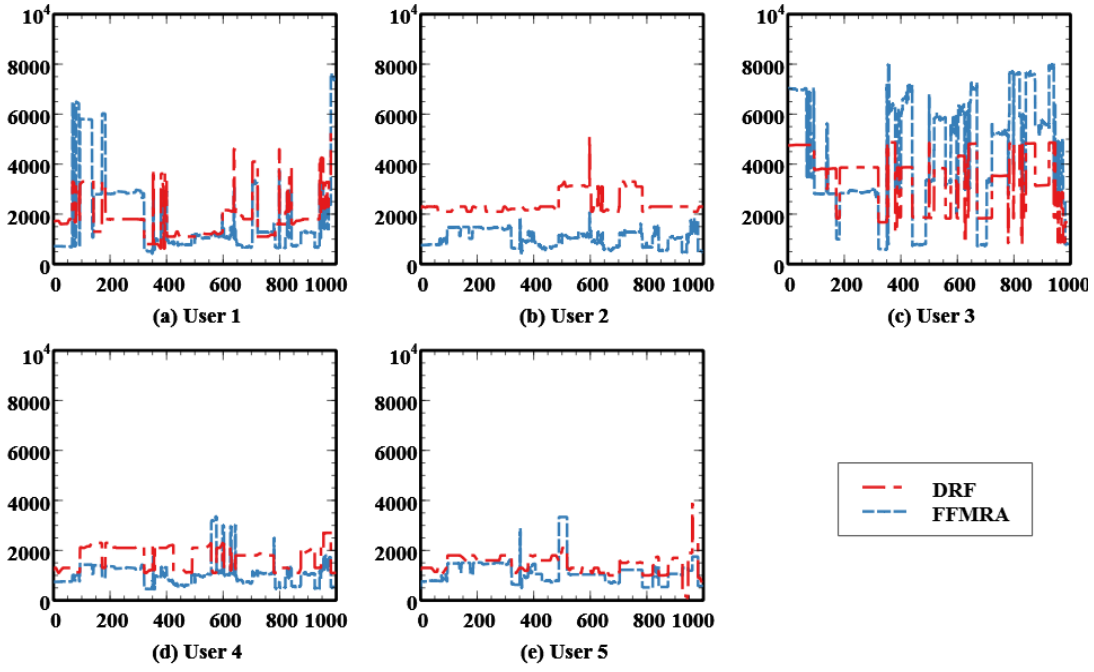


FIGURE 7.15: Allocated RAM for all users in DRF and FFMRA

Figure 7.16 describes disk allocation in DRF and FFMRA. Based on the figure, most of the users experience a variety of fluctuation in the different timetables. Meanwhile, user 3 has the lowest amount of allocation on disk, as there is no disk-intensive submission by this user. Hence, over the first 300 iterations, user 2 continuously submits disk-intensive tasks which cause maximum resource consumption by this user. In other words, commenting on data presented in the figure, user 1 peaks its highest allocation between time intervals 0, and 100. According to Figure 7.12(c), user 3 is the only one who does not submit tasks with dominant resources over the disk. Similar to other resource types, the allocation for users with non-dominant tasks in FFMRA is almost always lower than DRF. The equal allocation for users with disk-intensive submissions is seen in Figure 7.14(b)(e) as over iteration 700, users 2 and 5 schedule an equal number of tasks with a dominant resource on disk under FFMRA. Also, in Listing 5.3, the whole

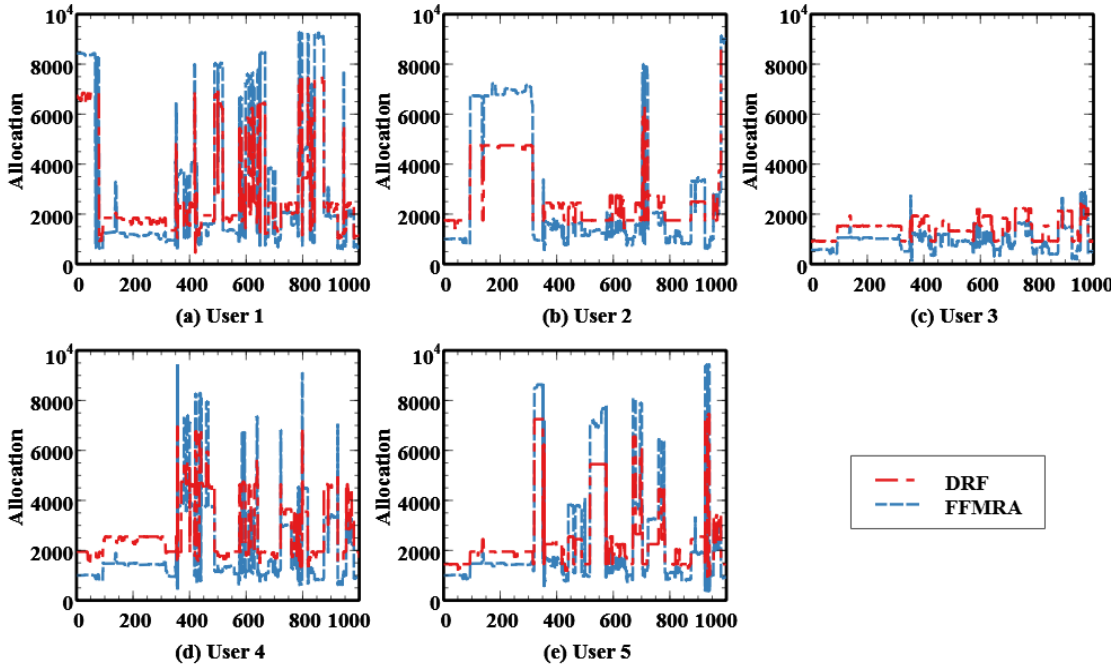


FIGURE 7.16: Allocated Disk for all users in DRF and FFMRA

allocation process over the first iteration is illustrated under the notion of FFMRA, considering requested and allocated resources.

TABLE 7.4

	User1	User2	User3	User4	User5
Requested CPU	372.659	3086.48	423.7	688.94	3099.16
Requested Memory	226.54	199.06	250.01	102.77	300.0
Requested Disk	402.96	450.0	12.52	382.6	450.0
Proportion capacity for dominants	30261.766938835575				
Proportion capacity for non-dominants	11738.233061164425				
Proportion for dominant CPU	14410.36520896932				
Proportion capacity for non-dominant CPU	5589.634791030679				
Proportion for dominant RAM	7205.18260448466				
Proportion for non-dominant RAM	2794.8173955153393				
Proportion for dominant disk	8646.219125381593				
Proportion for non-dominant disk	3353.780874618407				

In Table 7.4, we selected one of the iterations in which the whole allocation process is obviously explained. As can be seen, there are five users that they request resources over three different resource types. Accordingly, users 2, 4, and 5 have submissions with a dominant resource on CPU while dominant resources for users 1, and 3 are disk, and RAM respectively. Since the summation of dominant resources is considerably higher than non-dominant shares, FFMRA considers a high proportion of resource pool for dominants. It yields a maximum allocation for example for user 1, having dominant

resource on disk storage. Although the demand is 402.01, however, as the proportion for the dominant disk is about 8646, user 1 receives the whole proportion as the only user with an intensive submission over disk storage.

7.3.2 Fairness

7.3.2.1 Fairness for group of users in each specific resource type

To evaluate the fairness, two methods are employed that were already discussed in 4.18 and 4.19. In a multi-resource environment, fairness could be defined in different aspects as it is an intuitive concept. However, fair evaluation methods such as *Jain's* index which is preliminary used in computer networks, could not be a suitable candidate to measure fairness in Cloud systems. In our proposed fairness evaluation approach, initially, it is necessary to make sure that the resource pool capacity is evenly shared among both groups of users with dominant and non-dominant demands on multiple types of resources. The main assumption is to divide the resource pool with the same ratio to groups of users with dominant and non-dominant resources on a specific resource type. For example, consider a server that hosts two, and three users with dominant resources on CPU, and RAM respectively. As a result, FFMRA assumes two groups of users, each of which contains users with a dominant resource on each particular resource type. Therefore, if 60% of the resource pool is exclusively assigned to one group, the same amount should be assigned to another group as well. This way of resource distribution leads to perfect complementary fairness as the β always tends to 0.

Figure 7.17 shows the fair distribution of resources over a set of groups along with demands, dominating on a particular kind of resource. As can be seen in the figure, in the iterations between 300 and 400 there is at least one user within each group with a dominant resource. Because of this, FFMRA considers around 70% of the resource pool to each of those groups. Moreover, the fair distribution is also observed in different percentages for all groups such as iterations $(400 - 500)$, and $(800 - 1000)$. In some circumstances, as there is no submission with a dominant resource on RAM, and disk, 100% proportion for dominants is assigned to the user with CPU-intensive task. The β index makes no sense in such scenarios with five users. In real-world scenarios, though, as the frequency of submissions with dominant and non-dominant resources are almost

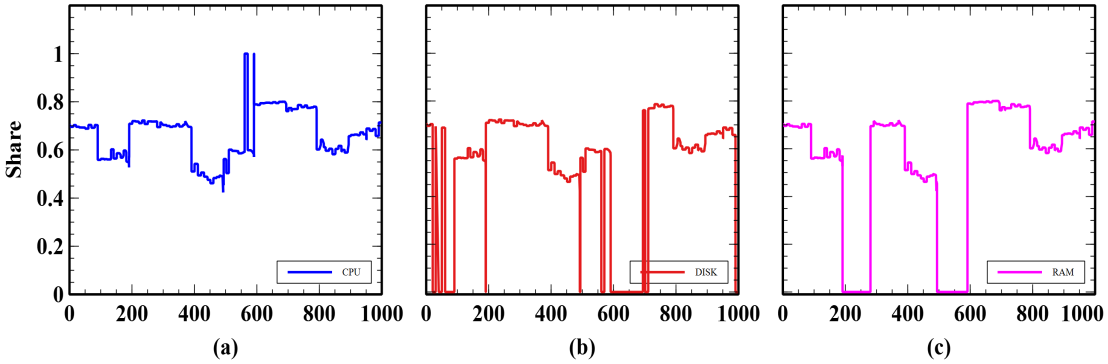


FIGURE 7.17: Sharing resource pool capacity among users with dominant resources

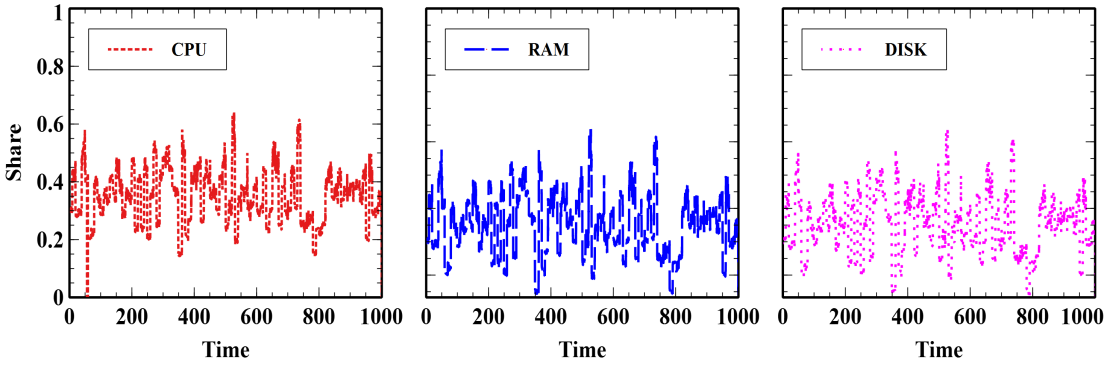


FIGURE 7.18: Sharing resource pool capacity among users with non-dominant resources

always positive, then $\beta = 0$. In terms of non-dominant resources, since the number of submitted tasks especially on RAM, and disk are mainly non-intensive, therefore, $\beta = 0$. Therefore, based on Figure 7.18, approximately in 90% of iterations the fair distribution is achieved.

7.3.2.2 Fairness for each individual user in each group in each specific resource type

In the previous section, the equal share of resource pool capacity among users with tasks dominated on a specific resource type is evaluated. In this section, the performance of FFMRA is determined among users with dominant resources in each group. To achieve this, the *Jain's fairness* index is employed as a generally accepted fairness evaluation method discussed in 2.2.5. In order to make sure that the fairness behavior of FFMRA is consistent, the same experiments in the previous section are taken into account, using 20 times experiments, and 100 users. Then, the average fairness measure is taken from all rounds of experiments.

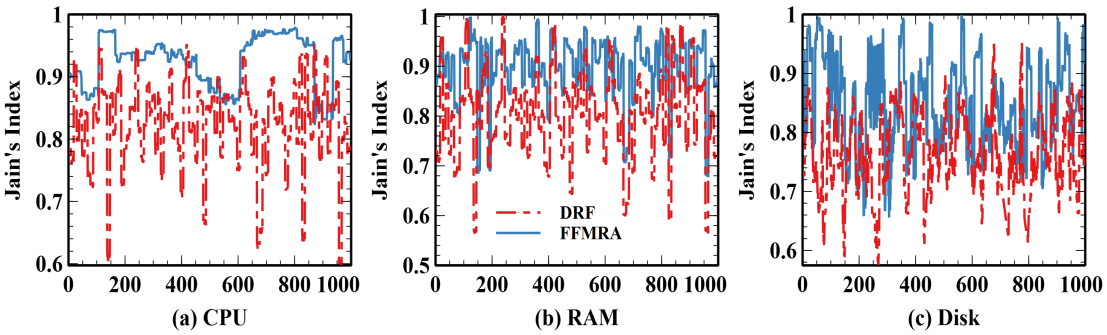


FIGURE 7.19: Fairness for users with dominant resources

Figure 7.19, depicts the comparison of FFMRA with DRF in terms of fairness with respect to three different resource types. As can be seen in Figure 7.19(a), FFMRA shows a great improvement in fairness compared to DRF. In fact, FFMRA is more consistent as it maintains fairness in a much higher value than DRF. It is obvious that the fairness index for DRF is dropped to 60% in some iterations. This is due to that, the requested CPU by users is dominant in a certain and fixed range of iterations. Therefore, as the allocated resources for each user is calculated by MLF-DRS policy, and also in most of the iterations, the requested resources are less than the *fairshare*, hence, the allocation for all users with the dominant resource is exactly the same. As a result, this way of allocation results in a high degree of fairness.

The fairness index for the RAM and disk is illustrated in Figure 7.19(b)(c). As the frequency of submissions with a dominant resource on RAM, and the disk is typically less than CPU, and also some users' demands with dominant resources are greater than the *fairshare*, consequently, the allocation is not identically the same in some iterations. This may result in degrading the value of the fairness index. Surprisingly, the fair value for the disk is better than CPU, and RAM as in a certain period of time the fairness index in disk allocation is near to 100%. Overall, these results have further strengthened our hypothesis that FFMRA provides considerable insight into the fairness issue in allocating multiple types of resources.

The total number of tasks that can be scheduled under FFMRA and DRF policies are analyzed. To achieve this, 10 times experiments are carried out over 1000 iterations. So, according to Figure 7.20, it is obvious that under FFMRA mechanisms, users can schedule noticeably much more tasks in contrast to DRF. This also confirms that not only FFMRA deals with fairness issue, but also provides efficient resource utilization.

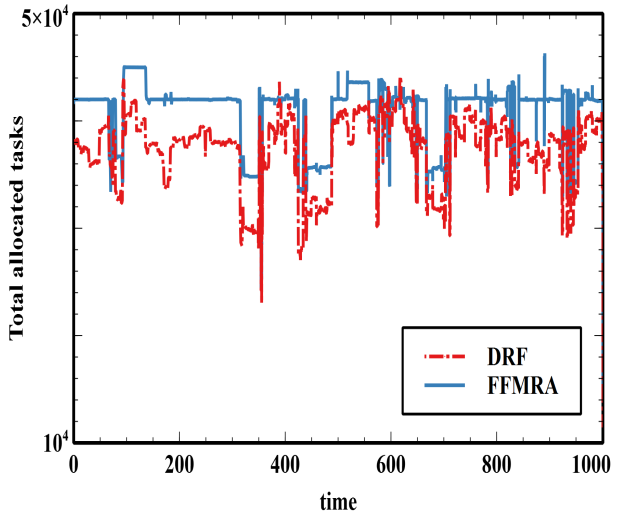


FIGURE 7.20: Total allocated tasks under DRF and FFMRA

7.3.3 Resource Utilization

In this section, the resource utilization of FFMRA is evaluated. The resource demands are randomly drawn from the stochastic workload generator using Google data traces patterns. Also, 10 times experiments are conducted over 1000 iterations. We then compare FFMRA with DRF by determining the average utilization from all rounds of experiments. The comparison of utilization in DRF and FFMRA is shown in Figure 7.21 for three different types of resources. In terms of CPU, and according to Figure 7.21(a), FFMRA shows a better resource utilization during all iterations. Moreover, CPU utilization is achieved approximately to 90% in a certain time series. Figure 7.21(b) depicts the utilization of RAM in both allocation policies. The figure clearly states that FFMRA gains a significant improvement on RAM utilization which is approximately 85-90%. On the contrary, the utilization achieved by DRF oscillates around 65-75%. With respect to Disk allocation, as can be seen in Figure 7.21(c), FFMRA shows significantly better utilization compared to DRF which utilizes around 70-73% of total disk capacity.

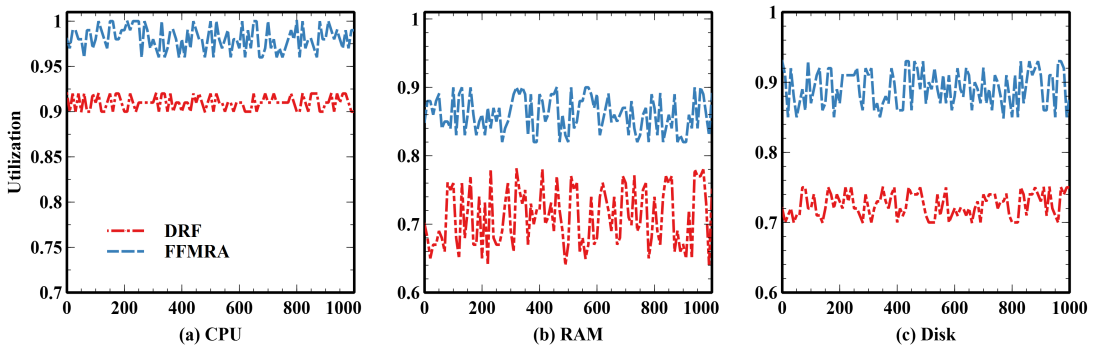


FIGURE 7.21: The comparison of resource utilization in DRF and FFMRA

7.4 Evaluations: H-FFMRA

This section evaluates the performance of H-FFMRA, considering resource allocation, utilization, and fairness. The experiments are conducted in the CloudSim as a simulation framework, driven by randomly generated workloads. In order for simplicity, two types of resources are taken into account that is CPU and RAM. Servers' configurations are arbitrarily selected to measure the actual functionality of H-FFMRA. Under random workload generation, users may represent various submissions, so that dominant and non-dominant resources may vary depending on their demands.

7.4.1 Resource allocation

The resource allocation under H-FFMRA is measured, taking into account four users, and 90 servers. All users are eligible to submit their tasks in all servers in absence of any placement constraint. In each iteration, users submit a variety of tasks, dominated either on CPU, or memory. Moreover, the allocation is examined over 1000 iterations. Figure 7.22, compares total allocated tasks in Multi-Host DRF, and H-FFMRA for four users. As can be seen in the figure, under H-FFMRA all users get approximately a total equal allocated task, as all resources are distributed fairly among groups of users in each server. Surprisingly, according to lines 15, and 16 in Algorithm 4, an absolute correlation is maintained for both dominant, and non-dominant resources in the entire system, as it also affects allocated tasks for all users. Figure 7.22(a) is the best representation of CPU allocation under H-FFMRA compared to Multi-Host DRF as all users in H-FFMRA get more resources than Multi-Host DRF. Furthermore, by iteration 800, all

users get nearly the same allocation in H-FFMRA. Nonetheless, after this point, as the dominant resources of submitted tasks are greater than the*fair-share* for some users, the allocation is slightly different. Therefore, some users get more resources than others. Typically, H-FFMRA attempts to capture an equal allocation for all users under a condition in which requested resources on dominant shares are less than the *fair-share*. Figure7.23 also illustrates RAM allocation in both approaches. The allocation under HFFMRA is significantly higher than multi-host DRF. After iteration 700, the allocated tasks for users in both algorithms start to go down. This is due to that the resource shortage in the system as at this point no more resources are provisioned to servers. In Figure 7.24 the resource allocation is represented across 30 servers for four users. The resources are distributed among users in each server according to the fair share function associated with the MLF-DRS mechanism. So, H-FFMRA tries to maintain a balance in allocating resources among all servers within the cluster. In almost all servers, the CPU is distributed evenly among users in all servers. This is due to that the requested resources from users are less than the fair share. This is a note that based on Figure 7.24, although the allocation for some users is greater than other users in a specific server, the distribution is considered evenly among users with different types of dominant resources.

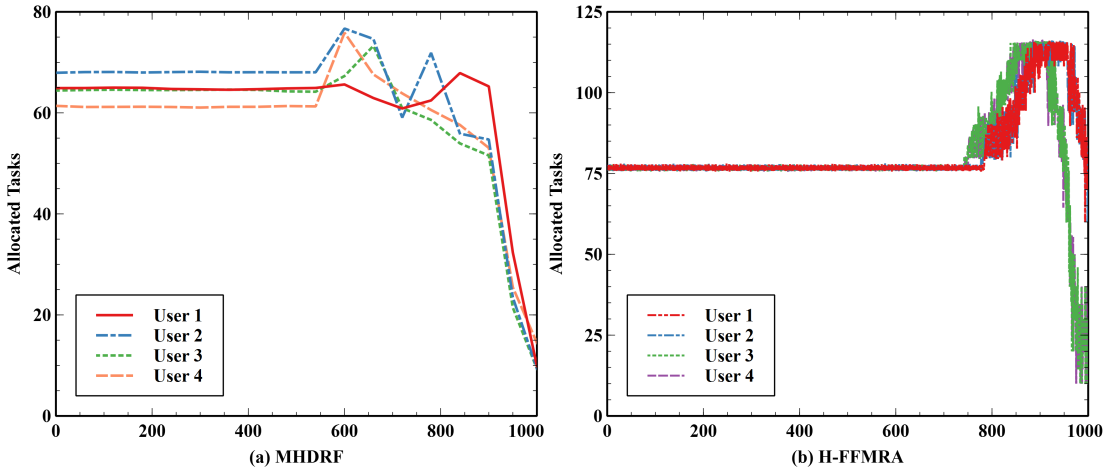


FIGURE 7.22: Allocated CPU to all users

7.4.2 Resource utilization

This section compares resource utilization under H-FFMRA and DRF policies, considering 2000 servers and 200 users. The configuration of servers are randomly selected so that the total capacities of CPU and RAM in the entire system are 2×10^6, and 4×10^6,

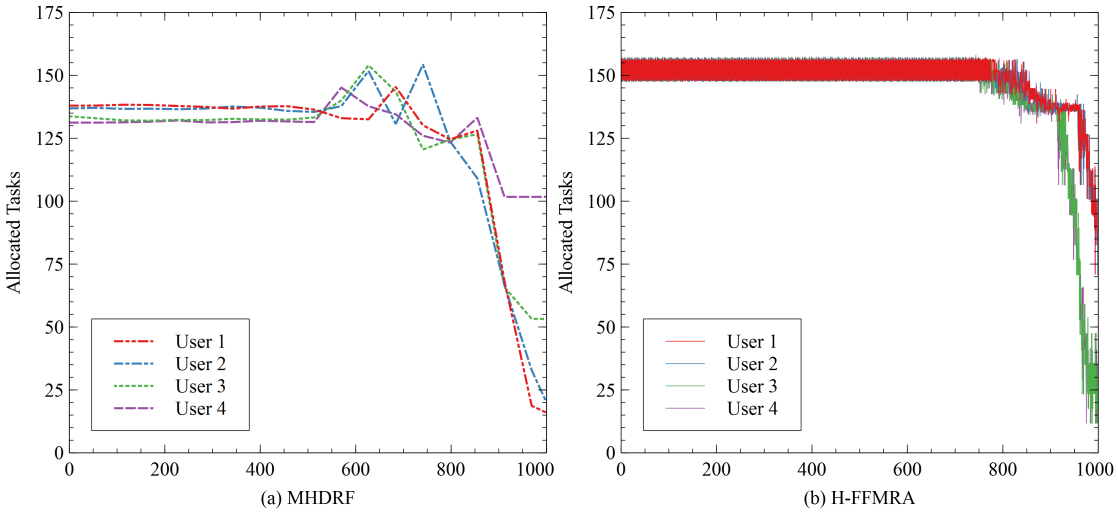


FIGURE 7.23: Allocated RAM to all users

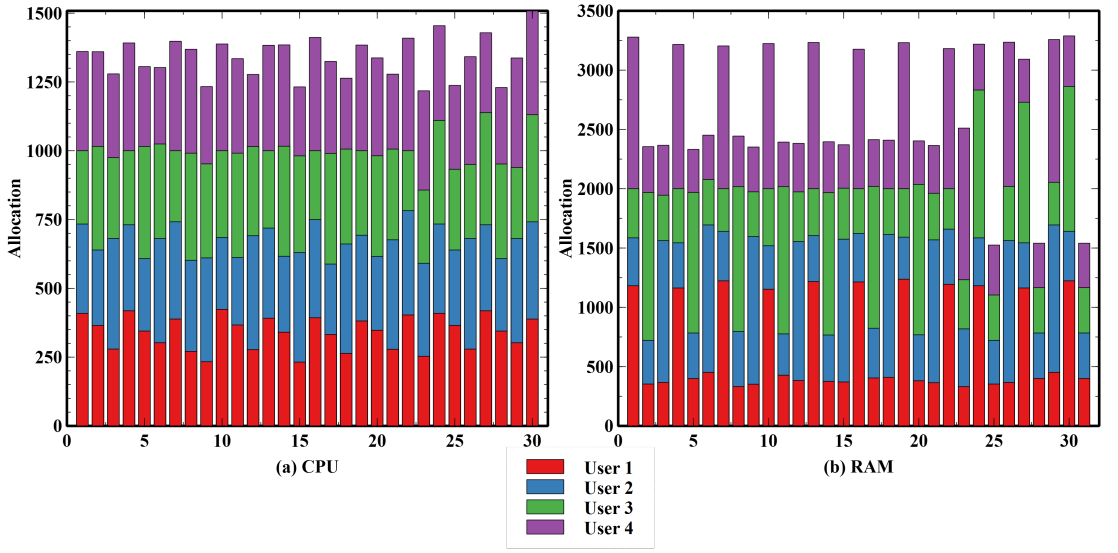


FIGURE 7.24: Allocated resources for four users under H-FFMRA policy over 30 servers

respectively. Then, the average resource utilization is determined in all servers in each iteration based on the following equation:

$$U_{s,k}(t) = \sum_{s=1}^{m} \sum_{i=1}^{n} \Pi_{i,s}^{k} / \Psi_s, s \in S \tag{7.8}$$

Where Ψ_s denotes the total number of servers in the system. The average utilization is a time-dependent function that captures the utilization in a specific range of iterations. So, if it is assumed that the time series in a range (t_0, t_{h+1}), then $U_{s,k}(0)$ gives utilization in average at iteration 0.

Figure 7.25 represents the resource utilization in both approaches in a time series experiment over 2000 iterations. Figure 7.25(a) depicts CPU utilization. As can be seen in figure H-FFMRA shows better CPU utilization than DRF in the multi-host setting. This is obvious in allocated tasks to users in Figure 7.22, as all users under H-FFMRA schedule significantly more tasks than DRF. Concerning RAM utilization, the superiority of H-FFMRA is seen under H-FFMRA in Figure 7.25(b). Typically, the DRF fails to satisfy the full utilization of resources, at least on a specific resource type in the heterogeneous resource profile. Accordingly, the recent developments of DRF from a multi-servers perspective unable to capture the full utilization of resources. This is worth mentioning that in almost all approaches, the CPU utilization is near to optimum, however, other resource types such as RAM, and disk may not have been fully utilized under those mechanisms. So, Figure 7.25(b) shows that H-FFMRA performs extremely better than DRF in consuming system resources. Except for resource shortage in some iterations, H-FFMRA obtains approximately a full utilization of resources compared to DRF in a multi-host setting.

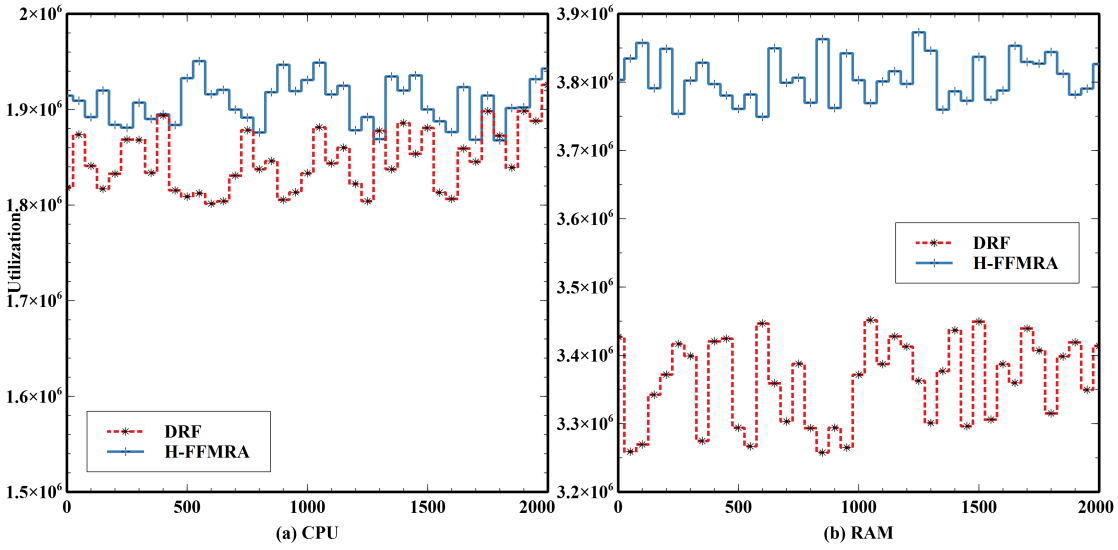


FIGURE 7.25: resource utilization in H-FFMRA, and MHDRF

7.4.3 Fairness

In this section, the performance of H-FFMRA is analyzed, considering fairness in server and user levels. For the first set of experiments, we measure the fairness using the $\beta - fairness$ index, then the $Jain's$ index is applied to evaluate the fairness for all users in the entire system. The experiments are conducted using time series over 1000

iterations, and 2000 servers so that the fairness becomes meaningful. Similar to the previous experiments, 200 users are considered with submitted various tasks. Then, the average fairness in each iteration is reported in two hierarchical levels.

7.4.3.1 $\beta - fairness$

Figure 7.26, illustrates the fair distribution of CPU, and RAM under H-FFMRA among all servers, and groups of users with dominant resources. As an example, in a specific range of iterations like $(20, 140)$, when 0.59 of RAM is considered on average for all servers with a set of dominant resources, the same amount for CPU is also considered. The figure states that a fully fair distribution is achieved in almost all iterations. In some points, the distribution is not the same for CPU, and RAM in some specific ranges, for example, $170 - 270$, $600 - 830$, and $840 - 900$. However, as there is a small deviation between $(0.62, 0.64)$, it is trivial in such a large-scale scenario. Overall, H-FFMRA achieves a fully fair distribution of resources among servers.

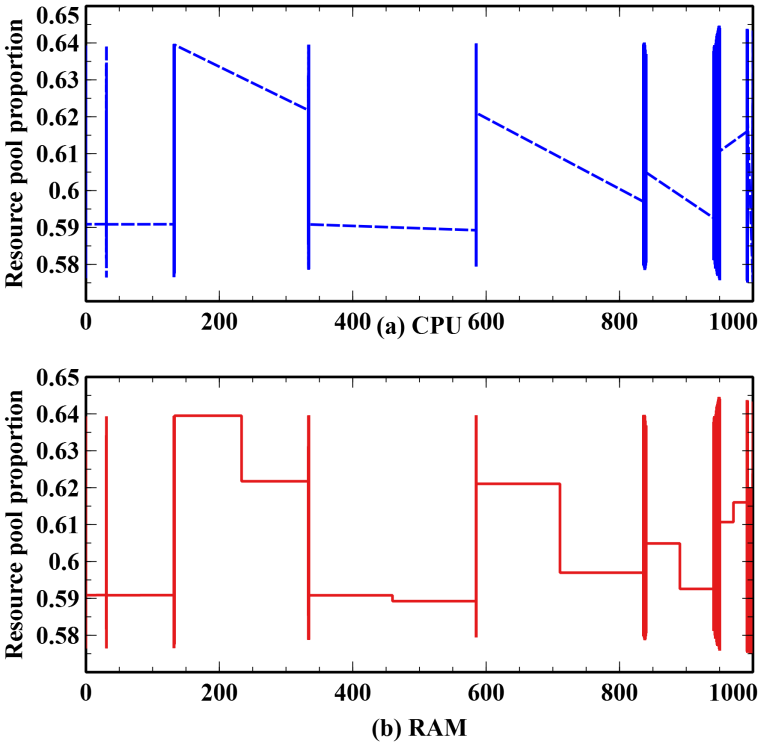


FIGURE 7.26: The fair distribution of resources among servers, considering dominant resources

In Figure 7.27(a)(b) the equal distribution of resources among servers is examined. Now considering non-dominant shares, the figure strongly confirms that the fair distribution

is well-achieved under H-FFMRA. Therefore, the values for β are exactly the same for both types of resources as the difference between the CPU, and RAM shares are about 0 in each iteration according to 4.43, and 4.45. So, based on the results in this section, fairness is captured in the first hierarchy of H-FFMRA.

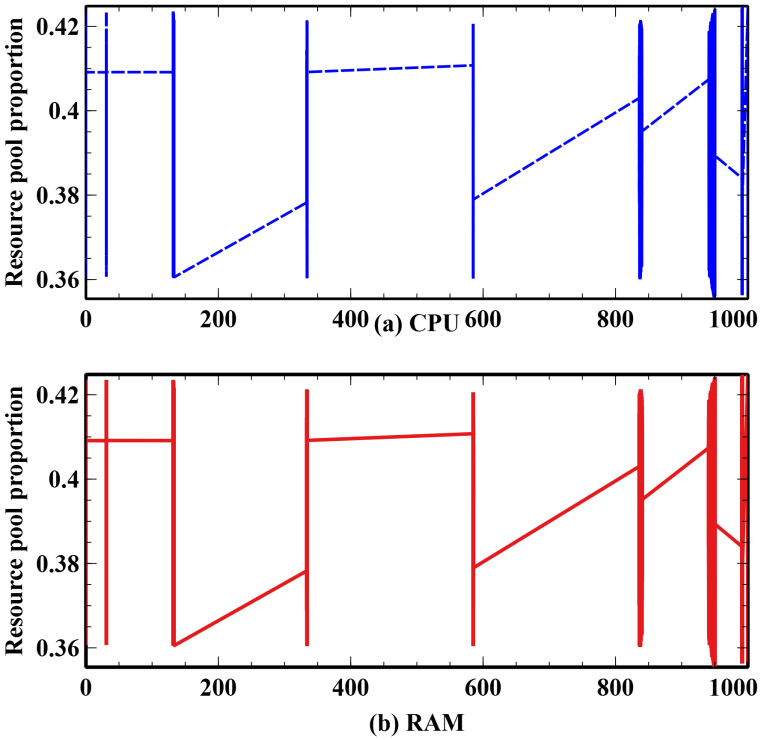


FIGURE 7.27: The fair distribution of resources among servers, considering non-dominant resources

Given the results in Figure 7.26, and 7.27, the beta fairness for dominant and non-dominant resources according to 4.43, and 4.45 is illustrated in Figure 7.28. Accordingly, it is the difference between the proportional allocation for groups of users with dominant, and non-dominant shares in CPU, and RAM.

As can be seen in the figure, the $\beta - fairness$ is satisfied for dominant, and non-dominant shares. Although there is a small variation for dominant shares in some iterations, it is trivial to consider the difference in the range $(0, 1)$. For example, if we consider the time series in a range $(150, 300)$, the difference is $0.64 - 0.62 = 0.02$ which is a very small value. So, the results confirm that H-FFMRA satisfies a fully fair distribution of resources in the first level of hierarchy. In the next section, we evaluate the fairness by applying $Jain's index$ to see how H-FFRMA behaves fairly among users concerning each specific server.

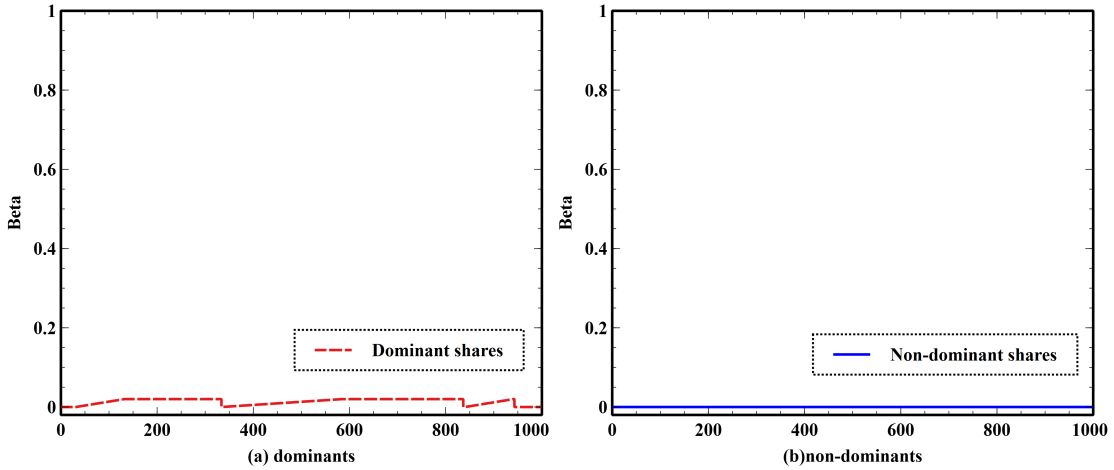


FIGURE 7.28: Beta fairness for dominant, and non-dominant shares

7.4.3.2 Jain's fairness index

In the second hierarchy of H-FFMRA, we examine the fair allocation concerning each server, and for each user in a specific group, considering dominant, and non-dominant shares. In this stage, a series of experiments are conducted, taking into account 400 users, and 3000 servers, over 1000 iterations. Figure 7.29, depicts fair CPU, and RAM allocation to users with dominant shares under H-FFMRA, and DRF. As the requested resources by users with submissions dominated on CPU and RAM could be greater than the fair share, the value for the Jain may be oscillated, and far away 1. So, based on the randomly generated workloads, in some cases, the allocation is not the same for users with dominant resources. Due to this, as can be seen in Figure 7.29(a), the fairness in CPU goes beyond 1. In terms of RAM, according to Figure 7.29(b), the fairness index oscillates between 0.95, and 0.96 as the requested RAM by users is greater than the fair share in a certain time series. On the other hand, in some iterations, Jain's index is 1 which denotes that almost all requested resources are less than the fair share.

In addition to dominant shares, Jain's index is determined for non-dominants as well. According to Figure 7.30, H-FFMRA performs remarkably fairer than DRF in multiple hosts, as it equalizes only dominant resources. Consequently, under DRF, non-dominant resources do not receive an equal share of the entire resource pool. Hence, the fairness index for H-FFMRA is closer to the optimal value of 1 for CPU and RAM compared to DRF. In a certain range of time series, the value of Jain's index is approximately 0.95. This is due to that in some cases the dominated demands on CPU or RAM are greater

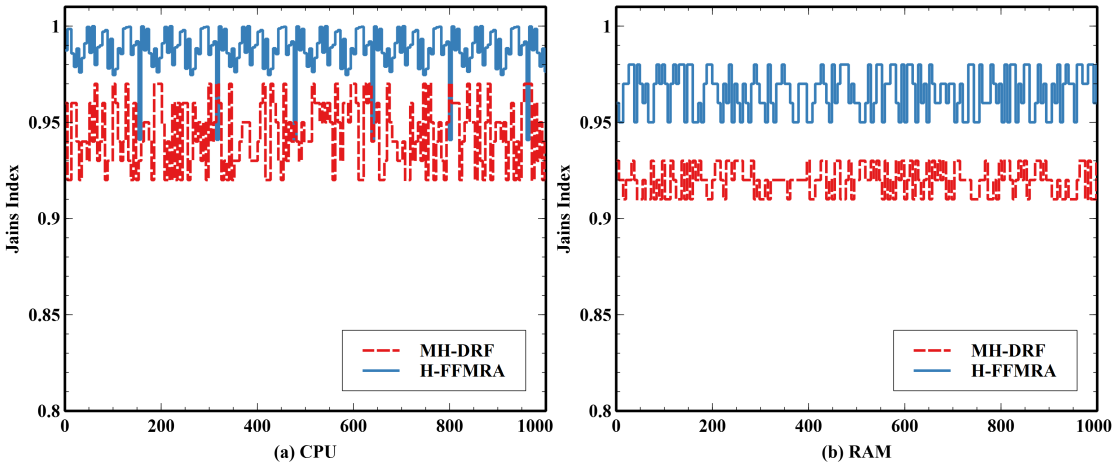


FIGURE 7.29: The Jain's index for users with dominant resources

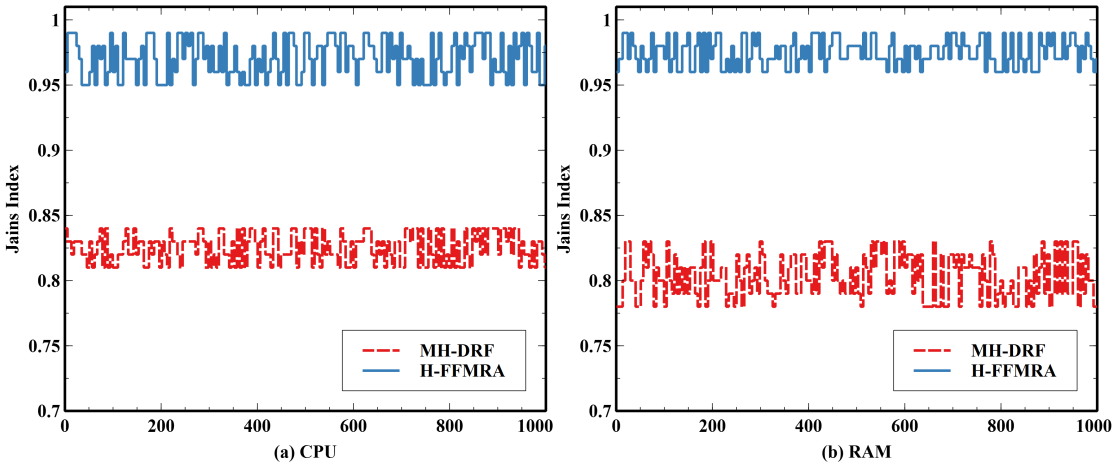


FIGURE 7.30: The Jain's index for users with non-dominant resources

or less than the *fairshare*. Therefore, similar to dominant resources, the allocation may not be the same for all users. Overall, the results verify that H-FFMRA is a fully fair allocation mechanism that captures the notion of intuitive fairness. This is worth mentioning that by increasing resource capacities for all servers and without having resource shortage, the possibility of achieving a fully fair allocation is considerably high.

7.5 Evaluations: MRFS

To evaluate the performance of MRFS, we use a system with Intel core-i5, 8250U, 1.6GHz CPU, and 8GB RAM. To conduct the experiments, we use CloudSim plus (Campos da Silva Filho et al. 2017), and CLoudsimRDA (Taddei 2015) simulation frameworks driven by the Google traces, and Randomly generated workloads in 1000 time-series experiments. we take into account different parameters such as the distribution of dominant resources in servers, proportion of resource pool among dominant and non-dominant resources, proportion of dominant resources for each specific resource type, and total allocated resource to users. During the experiments, each server is configured with $< 3000CPU, 6000RAM >$ since the total capacities of the data center are $< 9000CPU, 18000GBRAM >$, and $< 26000CPU, 52000GBRAM >$. Also, the demand profiles are normalized by $1/r^k_{i,s}$.

Figure 7.31 illustrates the number of dominant resources in each server in 1000 iterations over three hosts. For example, based on Figure 7.31(4)(5) after iteration 900, the difference of dominant resources in servers A, and B is increasing. However, as it is shown in Figure 7.31(6), MRFS tries to minimize it in server C. Also, as can be seen in Figure 7.31(1)(2)(3) the population of dominant resources in CPU is significantly higher than dominated tasks on RAM. However, under MRFS scheduling as it is illustrated in Figure 7.31(4)(5)(6), the difference between the number of dominant resource types is considerably low. This is due to that under the MRFS policy, the scheduler tries to equalize the number of dominant resources in each server in the entire data center.

Equalizing the number of dominant resources for each specific resource type may contribute to maximizing the proportion of the entire resource pool for all dominant resources. To perform this, we conduct experiments with the same server configuration and taking into account the frequency of dominant resources in Figure 7.31. Accordingly, we compare the allocation under H-FFMRA in the presence, and absence of MRFS. Figure 7.33(a)(b) clearly presents that the proportion of resource pool for dominant resources under the H-FFMRA in presence of MRFS policy is considerably better than H-FFMRA in absence of it. However, Figure 7.32(b) presents that the proportion for non-dominant tasks on CPU, and RAM without MRFS is higher than scheduling under MRFS.

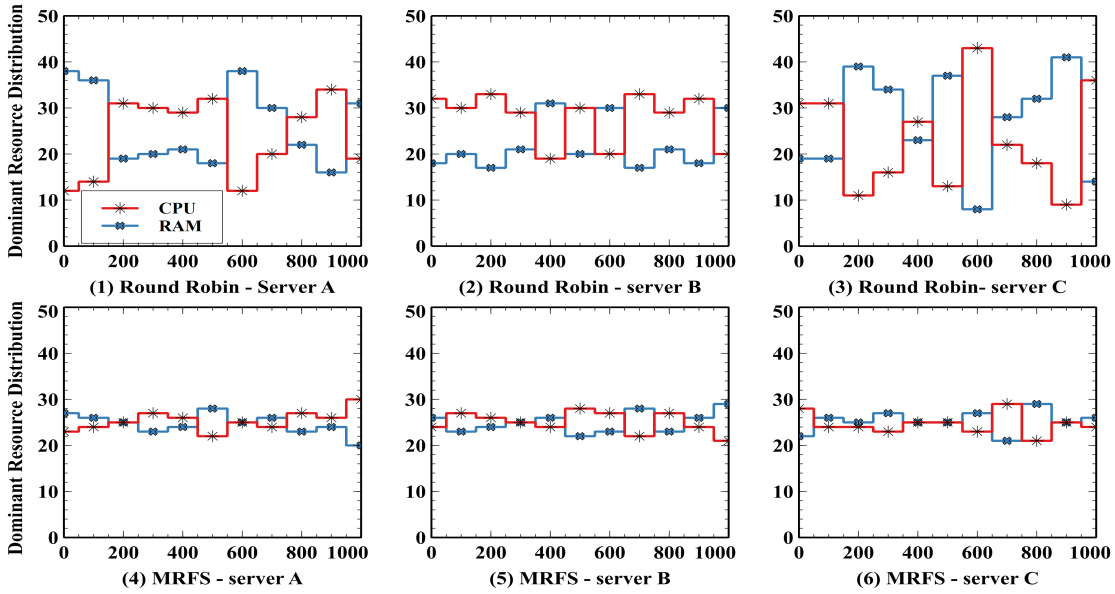


FIGURE 7.31: The number of tasks scheduled as dominant resources in three servers under Round Robin and MRFS policies

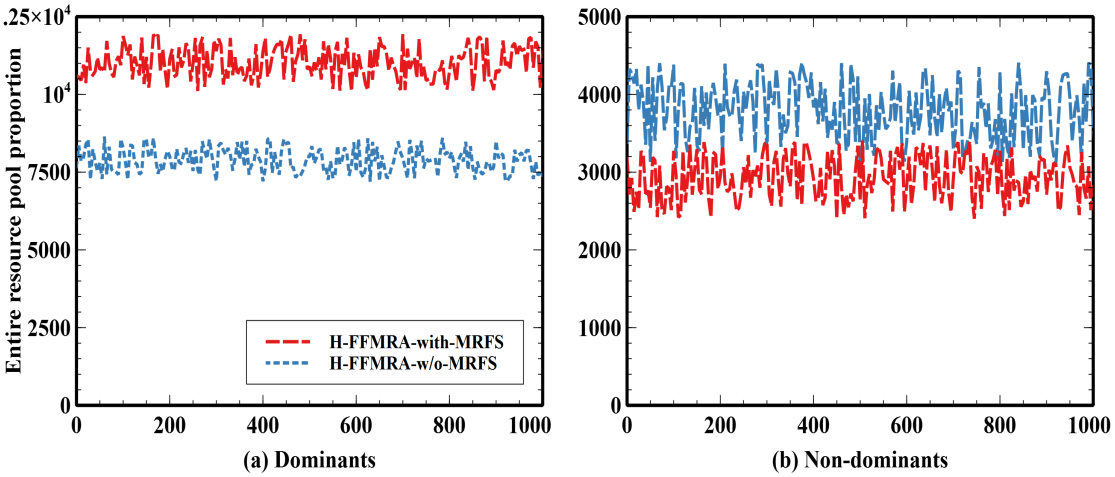


FIGURE 7.32: The proportion of entire resource pool for tasks with dominant and non-dominant resources in presence and absence of MRFS

Assume that without MRFS a certain proportion of data-center resources is allocated to each group of users with submissions, dominated on a particular resource type. Since, under MRFS, the number of dominant resources on CPU and RAM is well-balanced based on Figure 7.33, the corresponding proportional value is considerably high compared to the same scenario in absence of it. In Figure 7.33(a) the proportion for CPU is higher than the value for RAM under MRFS. This is due to that the sum of requested demands for CPU is greater than RAM.

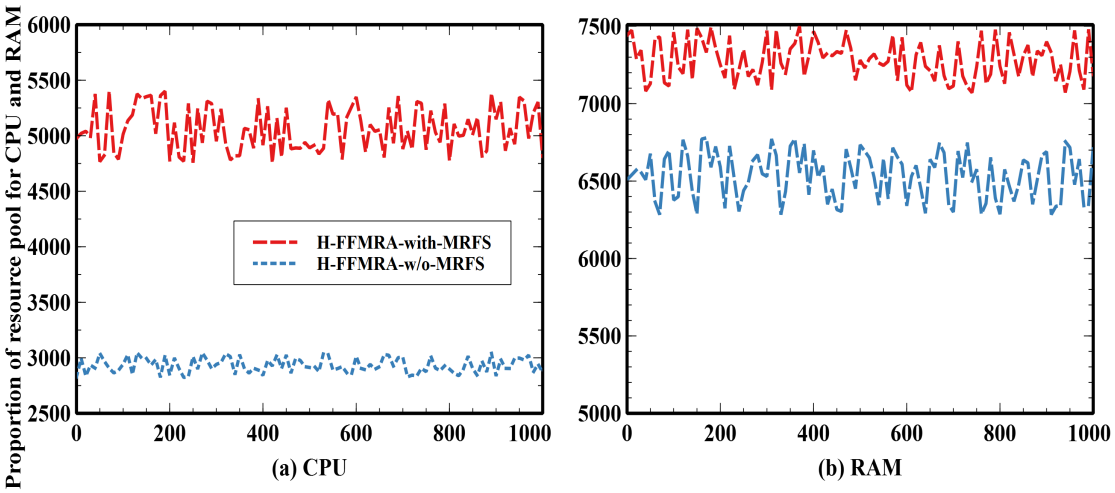


FIGURE 7.33: The proportion of entire resource pool, allocated to tasks with dominant and non-dominant resources

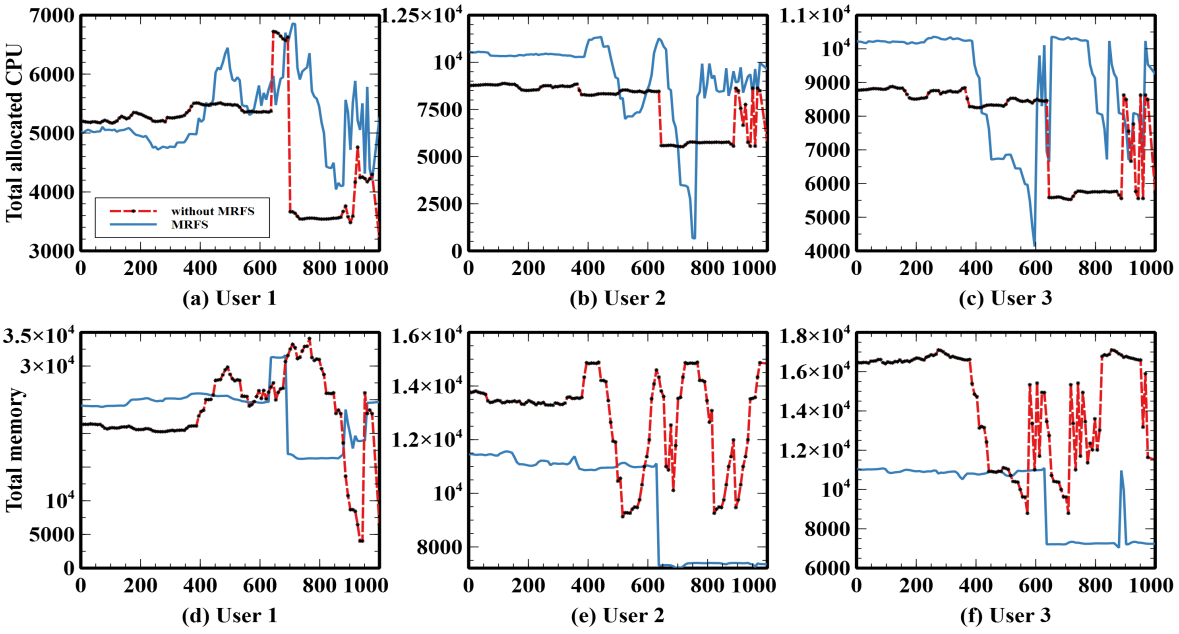


FIGURE 7.34: Total allocated resources to users across servers with and without MRFS

7.5.1 Allocation

To examine the performance of MRFS in terms of resource allocation, we conduct 100 time-series experiments with arbitrarily generated demands. It is assumed that five users, each submits an identical number of tasks across 100 servers with a total capacity of $< 26000CPU, 52000GBRAM >$. Figure 7.34 represents the allocated tasks to users under MRFS. Typically, as the competition among tasks with a particular dominant resource is well-balanced in MRFS, the total allocated resources to users is slightly better compared to the scenario in absence of MRFS. However, based on Figure 7.34(1) without

considering MRFS, the system allocates more resources to user A due to that he/she has more non-dominant task submissions on CPU in contrast to users B, and C. On the other hand, users A and C regularly submit non-dominant tasks on RAM. Therefore, the allocated tasks to these users in some iterations are lower than the allocation under MRFS. The penalty variable α is shown in Table 7.6 where the values are randomly drawn from different servers in time-series experiments. Accordingly, the greatest value in the table refers to a higher penalty. As it has been already discussed, the scheduler applies α for each user's allocation based on 5.4. The calculations of α, δ-CPU, and δ-RAM are randomly drawn using β values for dominant, and non-dominant CPU and RAM that is represented in TABLE 7.5. This is a note that the values in both tables are strongly dependent on each other. The Figure 7.36 represents the variations in β values for dominant and non-dominant CPU and RAM in 1000 iterations. For example, based on Figure 7.37(a)(b) and over iteration 250, the difference between δ for CPU, and RAM is a very small value. Consequently, the penalty value α is a small amount according to Figure 7.37(c). This is a note that based on 7.36, the difference between δ for dominant and non-dominant CPU and RAM play a direct role in determining the value for Λ, and α. Hence, the minimum difference in the same iteration between Figure 7.36(a)(c) leads to a minimum value for δ, and α.

```
for(double d:domcpulist1.values()) {
        sumaggdomcpu+=d;
        }
        for(double d:domdisklist1.values()) {
        sumaggdomdisk+=d;
        }
        deltaCPU=sumaggdomCPU/domcpulist1.size();
        deltadisk=sumaggdomdisk/domdisklist1.size();
        lambdaCPU=deltacpu/CPUCapacity;
        lambdastorage=deltadisk/Storagecapacity;
        alpha=Math.abs(lambdaCPU-lambdastorage);
```

7.5.2 Sharing incentive

To analyze the sharing incentive property under MRFS policy, we conduct an experiment using 100 servers, each of which is eligible to host 40 demands at time t. It is also assumed that the configuration of all servers is identical. Therefore, each task is recognized by a global dominant resource on a particular resource type. Furthermore, we have selected

TABLE 7.5: The value of β in randomly selected iterations for tasks with dominant and non-dominant resources

Delta dom-cpu	Delta dom-RAM	Delta non-dom-CPU	Delta non-dom-RAM
1.066834069	0.728409041	0.108561842	0.250502592
1.035760848	0.719698324	0.10183153	0.244452501
1.012268025	0.711362962	0.103893504	0.246280228
0.999276224	0.700556703	0.107685474	0.262689845
0.998897746	0.820490563	0.141714168	0.312789566
1.066772968	0.728390924	0.108579392	0.250518144
1.035780524	0.719710625	0.101816593	0.244440247
1.012274786	0.711355004	0.103900048	0.246288731
0.999298437	0.700558314	0.10768599	0.262690995
0.998866481	0.820515924	0.141721611	0.332773318
1.066752938	0.728385349	0.108584972	0.250522867

TABLE 7.6: Determining α derived by the value of δ from randomly selected iterations

Lambda CPU	**Lambda RAM**	**Alpha**
2457.352941	10203.4375	0.264436581
2600	10289.0625	0.254453125
2705.588235	9435.625	0.201222426
2720.882353	9909.375	0.223380515
2173.333333	8273.888889	0.196361111
2457.255882	10203.25	0.264436912
2600.264706	10289.27188	0.254437123
2705.411765	9436.03125	0.201260386
2600.343791	10289.34784	0.254433013
2705.359572	9436.175506	0.201272818
2720.809276	9909.384329	0.223388289
2173.430079	8273.062114	0.196310098

eight demands in random from one of the servers. According to Figure 7.35(a) all demands schedule more CPU-dominated tasks than the *fairshare*. On the other hand, and based on Figure 7.35(b) demand 5 is unable to schedule more tasks than the *fairshare* in some iterations. This is due to that the scheduler fails to find a suitable host to locate that task. Consequently, demand 5 is placed in non-dominant queues as it receives less amount of resource than the *fairshare*. However, this is trivial and it could not happen in scenarios with a large number of servers. Nonetheless, in a general setting, MRFS captures $\%95-97$ sharing incentive for demands, dominated on a specific resource type. For example, in a scenario with 10000 servers, the probability of locating a task with a dominant resource in the non-dominant queue is something like 1/10000 which is a trivial value. Therefore, on this occasion, the sharing-incentive property is almost fully achievable.

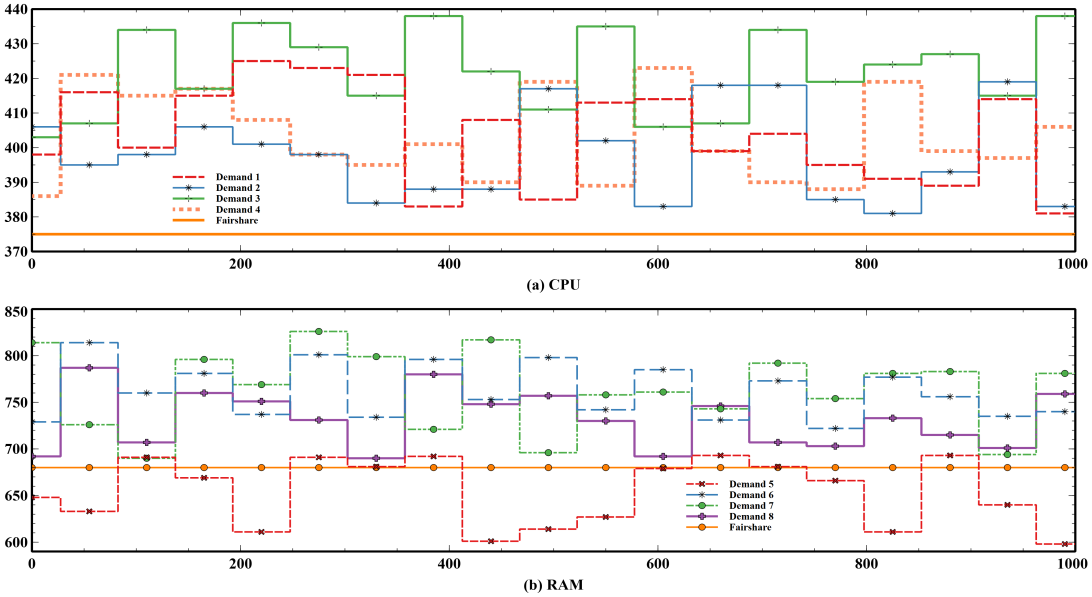


FIGURE 7.35: Total allocated tasks for eight incoming demands under MRFS policy

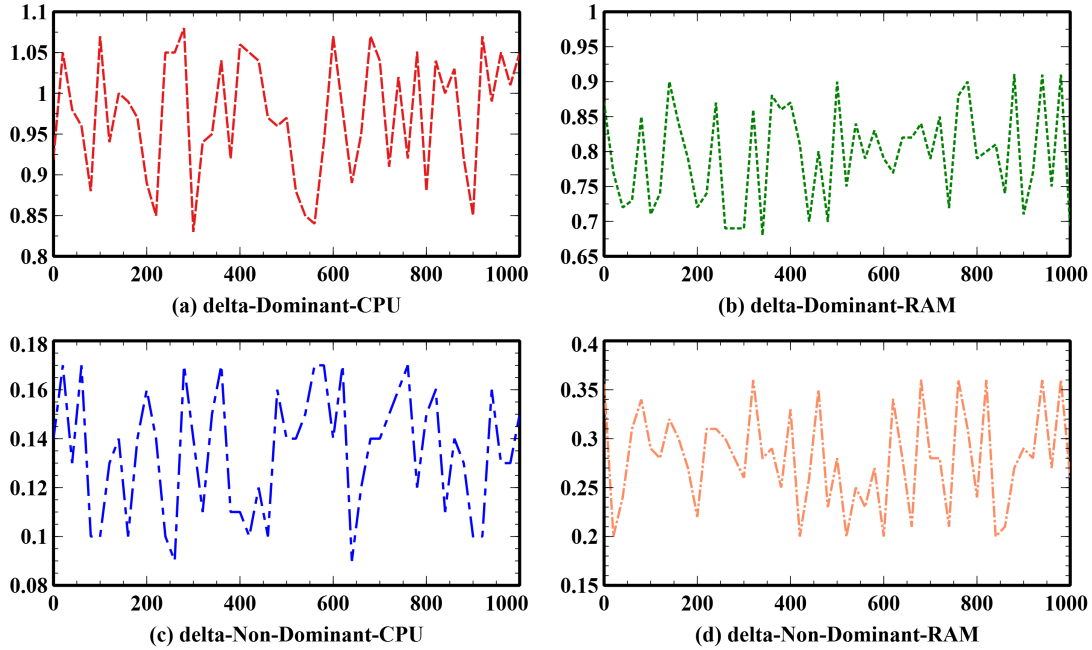


FIGURE 7.36: The values for δ for submissions with dominant and non-dominant resources

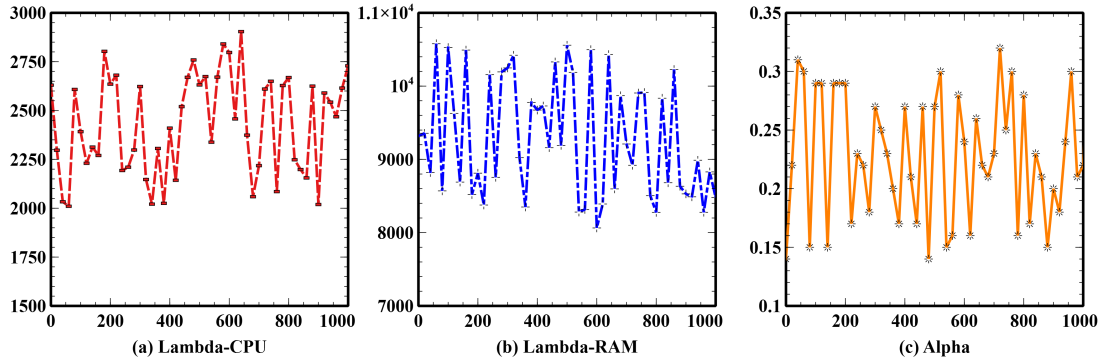


FIGURE 7.37: The values for Λ and α in 1000 iterations

7.5.3 Pareto efficiency

To evaluate the Pareto efficiency under the MRFS mechanism, we conduct eight experiments over 1000 iteration, taking into account servers with diverse configurations in terms of capacity. Three users are considered to examine how Round Robin behaves against the MRFS scheduling policy. We introduce a new metric to examine the Pareto-efficiency for each user which is determined in 7.9. To achieve the Pareto-index value as it is shown in Figure 7.38, we define a new measure as follows:

$$P_e = \Pi_i^{max} - f_k \tag{7.9}$$

Where P_e indicates the *Pareto-index* which is the difference of the maximum allocation Π_i^{max} of a resource, dominated on a specific resource type and the *fairshare* f_k. P_e denotes a perfect Pareto efficiency if $P_e \implies 0$. In other words, any value of P_e towards 0 is interpreted as a perfect Pareto efficiency. According to Figure 7.38(a)(b)(c), the Pareto index is determined for three users over many servers. Based on the figure, all users benefit from a higher degree of Pareto efficiency under the MRFS scheduling policy compared to allocation under the Round Robin mechanism. Also, as can be seen in the figure, since the number of servers is increased, the Pareto-index is improved and goes toward 0. This is due to that MRFS easily finds those servers with maximum resource availability and an equal number of tasks dominated on a particular resource type. This is worth mentioning that in some configurations, the Pareto-index for a configuration with a large number of servers is not improved compared to a minimum number of servers. For example, in Figure 7.38(c), the Pareto index is increasing and goes beyond zero. This is because, in a high percentage of iterations, user 2's requested resources

are less than users 1 and 3. Accordingly, the total allocated tasks to user 3 are less than others which results in a degraded Pareto index. However, it is obvious that under MRFS the Pareto efficiency is improved compared to an allocation under Round Robin.

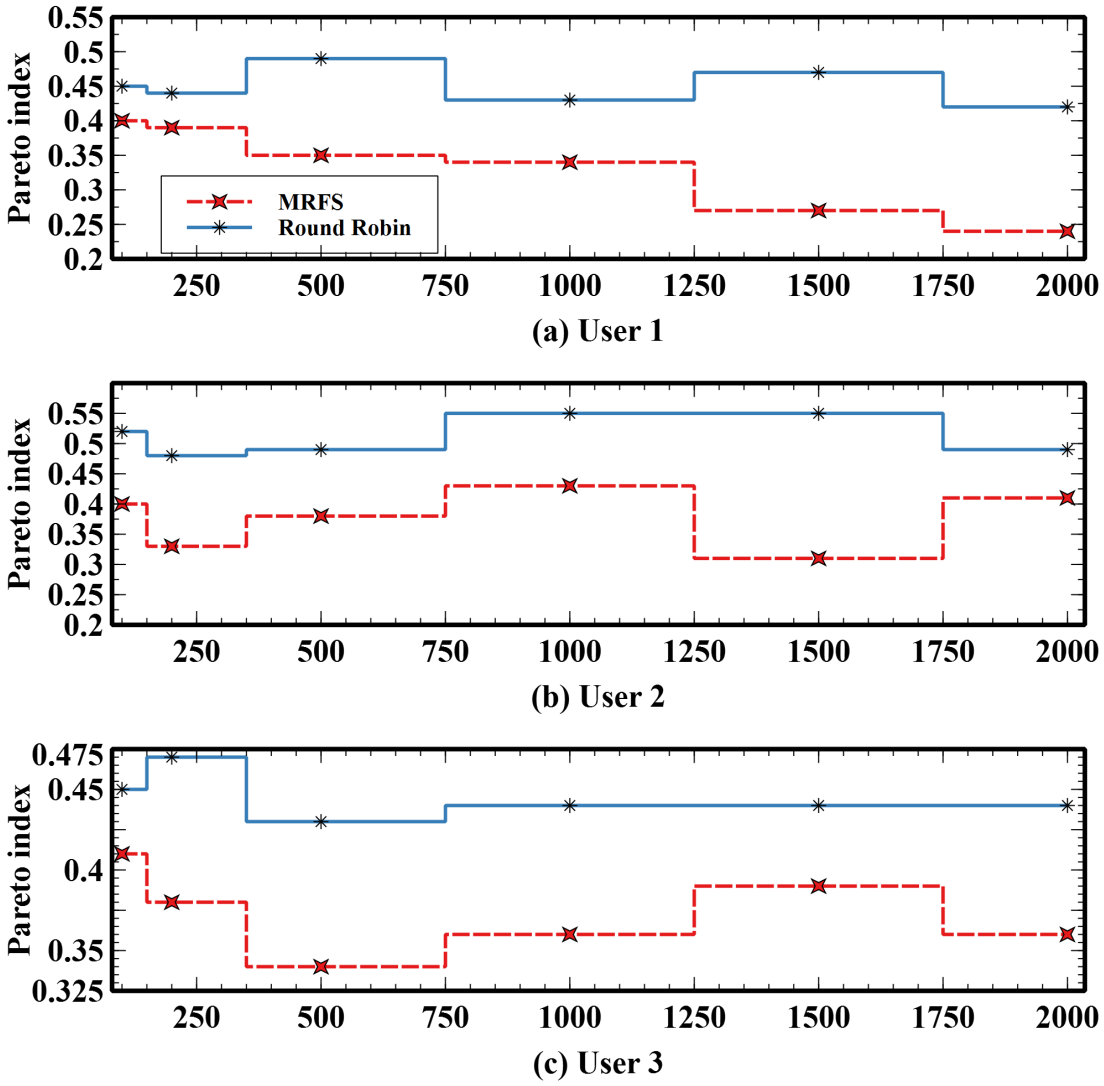


FIGURE 7.38: The index for Pareto-efficiency for three users over the servers with different resource capacities - The capacities are considered as total capacity including both CPU and RAM resources.

7.6 Summary

In this chapter, we evaluate the performance of proposed algorithms in the Cloudsim simulation framework. We consider different metrics such as fairness, resource allocation, utilization, and response time. We first investigate the workload generation, using mathematical formulations and presented a sample generated workload as incoming

tasks. The simulations driven by the randomly generated workloads and Google workload traces indicate that the proposed algorithms achieve better utilization and fairness compared to DRF.

In particular, We present that MLF-DRS achieves better fairness and utilization of resources, compared to DRF. We also acknowledge that the FFMRA achieves a fair distribution of resources among users, while the trade-off between fairness and efficiency is maintained. The H-FFMRA as the extension of FFMRA demonstrates promising performance in terms of fairness and efficiency compared to the DRF in a multi-server setting. We also examine our proposed β-fairness measure against FFMRA and H-FFMRA. Accordingly, the results confirm that under these mechanisms, the resources are allocated fairly among users with dominant and non-dominant shares. We further evaluate the performance of MRFS against the conventional Round-robin task scheduling mechanism. Accordingly, we test the functionality of MRFS in terms of resource allocation associated with H-FFMRA. We then conduct experiments in the absence and presence of MRFS policy. The results indicate that MRFS successfully equalizes non-identical dominant resources in each server. Besides, the experiments confirm that the sharing-incentive and Pareto-efficiency features were satisfied under MRFS policy.

Chapter 8

Conclusions

8.1 Summary

In this thesis, we study the resource allocation problem with fairness in Cloud Computing systems in presence of multiple types of resources, and servers. Since DRF was the first approach to address the fair resource allocation issue in the Cloud, different developments were proposed to deal with a number of drawbacks associated with DRF. Despite several fairness properties satisfied by those approaches, we identified a major gap in the literature that targets the notion of intuitive fairness and Pareto-efficiency. To deal with this problem, in the initial step we propose a new fair allocation algorithm called MLF-DRS which takes into account a fair share function to calculate allocations. MLF-DRS provides full utilization of resources as well as satisfying fairness properties. Then, we propose FFMRA as a fully fair allocation mechanism. FFMRA employs a two-level hierarchy model to allocate resources, employing the MLF-DRS mechanism. To extend FFMRA in multiple servers, H-FFMRA is introduced which inherits all existing features of FFMRA to allocate resources in heterogeneous servers. Then, we propose a Multi Resource Fair Task Scheduling mechanism called MRFS. MRFS tries to assign the most efficient servers to users' tasks while the equal number of dominant resources of each specific resource type is maintained as well as the maximum availability of resources. To evaluate proposed algorithms, we conduct time-series experiments in the CloudSim driven by randomly generated workloads. The evaluations represent that our proposed algorithms perform better than DRF in terms of utilization, and fairness. Furthermore, we investigate the applicability of proposed algorithms in Kubernetes as

a state-of-the-art containers orchestration platform. We propose a model that enables the Kubernetes scheduler to assign resource limits with respect to dominant resources which leads to minimizing the number of pod evictions.

8.2 Contributions

The main contributions of this thesis are categorized in the following sections.

8.2.1 A mutli-level fair allocation algorithm in the cloud

A multi-level fair resource allocation algorithm called MLF-DRS is developed with the assumption of a single-server environment in 4.2. Regarding this, a decision-making policy is demonstrated in terms of the boundary of demands, corresponding to the fair-share. We further considered the equalization of resources based on the fair-share and taking into account identical dominant resource types. Besides, to address the problem of response time for non-identical dominant resources, a new queuing mechanism is introduced. The experiments conducted in the CloudSim indicate that the MLF-DRS performs better than DRF in terms of fairness, resource allocation, and utilization. Moreover, MLF-DRS satisfies a number of fairness features such as envy-free, strategy-proof, Pareto-efficiency, and sharing incentive.

8.2.2 A fully fair multi-resource allocation algorithm in the cloud computing

We propose the FFMRA (4.3) as a fully fair resource allocation mechanism. FFMRA employs a two-level hierarchy strategy to ensure the fair distribution of resources among users. In the first step, it calculates the whole contribution of dominant and non-dominant resources in the entire resource pool. Then, it equalizes the aggregate shares, allocated to groups of users with dominant, and non-dominant resources. In the last step, the MLF-DRS is employed to calculate final shares. Furthermore, a new fairness measure (4.3.2.3) is introduced to judge fairness in multi-resource environments. The experiments driven by randomly generated workloads in the CloudSim confirm that not

only FFMRA achieves an efficient allocation compared to DRF, but also, it achieves a fully fair resource allocation. Also, FFMRA satisfied desirable fairness properties.

8.2.3 A Completely Fair Multi Resource Allocation Approach in Heterogeneous Servers

In H-FFMRA (4.4), we generalize the main intuition behind the FFMRA in the multi-server profile. Accordingly. the proportion of the entire resource pool is formulated in terms of Global aggregate dominant resource for dominant, and non-dominant shares. We then re-formulate the fairness measure, developed in FFMRA to judge the fair distribution of resources among groups of users with tasks dominated, and non-dominated on a particular resource type. Based on the randomly generated workloads, carried out in the CloudSim, our proposed mechanism captures intuitive fairness by evenly distributing resources among users according to the two-level hierarchy model in FFMRA. The experiments verify that H-FFMRA improves fairness by %20 for dominant, and non-dominant resources compared to Multi-Host DRF. Moreover, H-FFMRA presents the full utilization of resources in normal conditions. We also prove that H-FFMRA meets some of the good fairness features.

8.2.4 MRFS: A Multi-Resource Fair Task Scheduling in the Cloud Computing Systems

In 5, we propose a novel fair task scheduling algorithm called MRFS, considering the equalization of the number of non-identical dominant resources in a multi-server cloud. The problem is formulated, employing the Lagrangian multipliers with emphasis on maximizing per-user utility functions. we aim to solve the problem of sharing-incentive, and Pareto-efficiency in recent approaches. The experiments, driven by the randomly generated workloads and Google traces confirm that MRFS maximizes users' utilities approximately by %15-20. Furthermore, the experiments prove that under MRFS, sharing-incentive, and Pareto-efficiency features are improved.

8.2.5 A new Approach to Calculate Resource Limits with Fairness in Kubhernetes

In 4.5, the problem of fairness in Kubernetes is investigated. We show that if the fairness algorithms are properly integrated with this environment, the possibility of pod evictions could be decreased. Accordingly, a model-based architecture is developed, focusing on integrating proposed algorithms. In this model, we empower the Kube-scheduler to take into account resource limits in scheduling decisions. We then restrict each pod corresponding namespaces to control the resource consumption based on the assigned resource limits.

8.2.6 Future work

In this thesis, we propose MLF-DRS, FFMRA, and H-FFMRA as the fair resource allocation algorithms along with MRFS as a new fair task scheduling mechanism. We also introduced a model to integrated fair resource allocation and scheduling algorithms in Kubernetes as a Cloud-native solution for container orchestration. The main focus of this thesis is to address the major gaps in these areas regarding intuitively fair resource allocation. For further developments, the proposed algorithms are expected to be cost-efficient as well. Hence, for future work, this capability would be considered for all these algorithms.

Also, in 4 and 6.3, we define mathematical and modeling representations to show that how proposed algorithms can be implemented in the Kubernetes framework. For the future direction, we will implement all our proposed mechanisms in BT's infrastructure.

Moreover, edge computing is considered to be the most challenging area concerning resource allocation and scheduling. While in this thesis the main cloud datacenter is taken into account, edge computing could open new challenges against recently proposed approaches. Task offloading from the main cloud data center and edge servers could be reduced by applying and developing proposed fair resource allocation and task scheduling algorithms in such a dynamic environment. Typically, in an edge environment, incoming workloads from IoT and edge devices are unpredictable. Therefore, to deal with such dynamicity, further, improvements are required to adapt proposed algorithms in edge infrastructure. Cloud-native solutions such as Kube-flow which is built on top of the

Kubernetes have been recently proposed to deal with machine learning workloads. The applicability of Kube-flow is highlighted in industry 4.0 applications. Accordingly, we will investigate the integration of our proposed algorithms with this promising extension of Kubernetes. Therefore, to make our algorithms adaptable in such an environment, we will consider autonomic computing principles.

Appendix A

MLF-DRS traces

```
 ========== RESULT ==========
VM ID    STATUS    ST        FT      Users
53       SUCCESS 0.01    165.23   user_53
29       SUCCESS 0.01    166.78   user_29
5        SUCCESS 0.01    166.84   user_5
17       SUCCESS 0.01    166.84   user_17
13       SUCCESS 0.01    167.05   user_13
9        SUCCESS 0.01    167.87   user_9
25       SUCCESS 0.01    168.06   user_25
1        SUCCESS 0.01    168.53   user_1
41       SUCCESS 0.01    168.7    user_41
57       SUCCESS 0.01    168.7    user_57
61       SUCCESS 0.01    168.71   user_61
45       SUCCESS 0.01    168.72   user_45
21       SUCCESS 0.01    169.07   user_21
37       SUCCESS 0.01    169.14   user_37
65       SUCCESS 0.01    169.62   user_65
33       SUCCESS 0.01    170.15   user_33
49       SUCCESS 0.01    170.5    user_49
0        SUCCESS 0.01    173.39   user_0
2        SUCCESS 0.01    173.39   user_2
3        SUCCESS 0.01    173.39   user_3
4        SUCCESS 0.01    173.39   user_4
6        SUCCESS 0.01    173.39   user_6
7        SUCCESS 0.01    173.39   user_7
8        SUCCESS 0.01    173.39   user_8
10       SUCCESS 0.01    173.39   user_10
11       SUCCESS 0.01    173.39   user_11
12       SUCCESS 0.01    173.39   user_12
14       SUCCESS 0.01    173.39   user_14
15       SUCCESS 0.01    173.39   user_15
```

```
16      SUCCESS 0.01    173.39  user_16
18      SUCCESS 0.01    173.39  user_18
19      SUCCESS 0.01    173.39  user_19
20      SUCCESS 0.01    173.39  user_20
22      SUCCESS 0.01    173.39  user_22
23      SUCCESS 0.01    173.39  user_23
24      SUCCESS 0.01    173.39  user_24
26      SUCCESS 0.01    173.39  user_26
27      SUCCESS 0.01    173.39  user_27
28      SUCCESS 0.01    173.39  user_28
30      SUCCESS 0.01    173.39  user_30
31      SUCCESS 0.01    173.39  user_31
32      SUCCESS 0.01    173.39  user_32
34      SUCCESS 0.01    173.39  user_34
35      SUCCESS 0.01    173.39  user_35
36      SUCCESS 0.01    173.39  user_36
38      SUCCESS 0.01    173.39  user_38
39      SUCCESS 0.01    173.39  user_39
40      SUCCESS 0.01    173.39  user_40
42      SUCCESS 0.01    173.39  user_42
43      SUCCESS 0.01    173.39  user_43
44      SUCCESS 0.01    173.39  user_44
46      SUCCESS 0.01    173.39  user_46
47      SUCCESS 0.01    173.39  user_47
48      SUCCESS 0.01    173.39  user_48
50      SUCCESS 0.01    173.39  user_50
51      SUCCESS 0.01    173.39  user_51
52      SUCCESS 0.01    173.39  user_52
54      SUCCESS 0.01    173.39  user_54
55      SUCCESS 0.01    173.39  user_55
56      SUCCESS 0.01    173.39  user_56
58      SUCCESS 0.01    173.39  user_58
59      SUCCESS 0.01    173.39  user_59
60      SUCCESS 0.01    173.39  user_60
62      SUCCESS 0.01    173.39  user_62
63      SUCCESS 0.01    173.39  user_63
64      SUCCESS 0.01    173.39  user_64

Resources:
mips: 599.53, ram: 499.44, bw: 452.79, disk I/O: 59.97
mips: 500.73, ram: 997.16, bw: 149.99, disk I/O: 50.24
mips: 604.02, ram: 497.03, bw: 120.95, disk I/O: 60.75
mips: 403.49, ram: 249.03, bw: 120.7, disk I/O: 40.25
mips: 608.31, ram: 499.39, bw: 455.87, disk I/O: 60.65
mips: 497.88, ram: 1000.64, bw: 149.72, disk I/O: 49.79
mips: 607.89, ram: 496.4, bw: 121.86, disk I/O: 60.62
mips: 401.63, ram: 251.87, bw: 119.51, disk I/O: 40.03
mips: 599.86, ram: 499.19, bw: 449.75, disk I/O: 60.15
```

```
mips: 498.1, ram: 999.08, bw: 150.09, disk I/O: 49.86
mips: 595.26, ram: 496.53, bw: 118.97, disk I/O: 59.63
mips: 404.98, ram: 250.67, bw: 121.13, disk I/O: 40.4
mips: 598.52, ram: 504.08, bw: 120.83, disk I/O: 60.39
mips: 604.67, ram: 500.33, bw: 452.1, disk I/O: 60.35
mips: 502.53, ram: 997.96, bw: 150.02, disk I/O: 49.91
mips: 607.51, ram: 493.98, bw: 121.51, disk I/O: 60.8
mips: 395.53, ram: 249.09, bw: 118.72, disk I/O: 39.79
mips: 604.95, ram: 498.36, bw: 453.15, disk I/O: 60.2
mips: 500.61, ram: 999.45, bw: 150.26, disk I/O: 50.34
mips: 593.25, ram: 502.83, bw: 118.61, disk I/O: 59.92
mips: 398.21, ram: 243.75, bw: 119.59, disk I/O: 39.78
mips: 608.29, ram: 499.3, bw: 456.88, disk I/O: 60.99
mips: 497.45, ram: 1001.95, bw: 149.86, disk I/O: 50.05
mips: 402.72, ram: 252.92, bw: 120.1, disk I/O: 40.09
mips: 605.84, ram: 497.13, bw: 120.81, disk I/O: 60.71
mips: 401.32, ram: 251.22, bw: 120.46, disk I/O: 40.37
mips: 606.45, ram: 504.99, bw: 456.15, disk I/O: 60.98
mips: 504.8, ram: 998.03, bw: 150.86, disk I/O: 49.98
mips: 599.09, ram: 497.39, bw: 119.95, disk I/O: 60.24
mips: 401.84, ram: 251.74, bw: 120.13, disk I/O: 40.24
mips: 595.14, ram: 505.31, bw: 445.76, disk I/O: 59.9
mips: 501.18, ram: 992.83, bw: 150.96, disk I/O: 50.35
mips: 592.92, ram: 497.34, bw: 118.41, disk I/O: 59.34
mips: 401.29, ram: 245.94, bw: 120.0, disk I/O: 40.19
mips: 606.74, ram: 502.63, bw: 453.92, disk I/O: 60.42
mips: 599.16, ram: 499.95, bw: 453.05, disk I/O: 60.35
mips: 502.04, ram: 997.49, bw: 150.27, disk I/O: 49.95
mips: 601.87, ram: 503.94, bw: 121.51, disk I/O: 60.28
mips: 400.37, ram: 252.63, bw: 119.21, disk I/O: 39.79
mips: 591.76, ram: 498.96, bw: 443.1, disk I/O: 59.47
mips: 503.54, ram: 1003.6, bw: 150.9, disk I/O: 50.05
mips: 613.66, ram: 496.61, bw: 121.58, disk I/O: 61.08
mips: 404.37, ram: 253.63, bw: 120.61, disk I/O: 40.2
mips: 610.02, ram: 499.26, bw: 455.54, disk I/O: 60.86
mips: 506.01, ram: 998.29, bw: 151.73, disk I/O: 50.6
mips: 497.27, ram: 998.54, bw: 149.66, disk I/O: 49.82
mips: 596.7, ram: 500.01, bw: 119.94, disk I/O: 59.94
mips: 404.96, ram: 244.64, bw: 121.28, disk I/O: 40.6
mips: 602.07, ram: 501.86, bw: 449.18, disk I/O: 60.14
mips: 492.91, ram: 995.47, bw: 148.42, disk I/O: 49.82
mips: 603.48, ram: 497.39, bw: 120.71, disk I/O: 60.15
mips: 403.7, ram: 251.83, bw: 120.87, disk I/O: 40.13
mips: 596.73, ram: 506.32, bw: 446.18, disk I/O: 59.63
mips: 502.49, ram: 999.14, bw: 150.48, disk I/O: 50.21
mips: 595.85, ram: 501.31, bw: 119.36, disk I/O: 59.75
mips: 399.23, ram: 256.14, bw: 120.18, disk I/O: 39.93
mips: 593.35, ram: 501.09, bw: 118.49, disk I/O: 59.11
```

```
mips: 596.93, ram: 498.52, bw: 449.87, disk I/O: 59.75
mips: 502.3, ram: 1003.63, bw: 150.15, disk I/O: 50.17
mips: 597.95, ram: 497.0, bw: 120.27, disk I/O: 60.14
mips: 393.89, ram: 247.01, bw: 118.19, disk I/O: 39.29
mips: 607.57, ram: 499.44, bw: 454.49, disk I/O: 60.45
mips: 504.26, ram: 999.2, bw: 151.37, disk I/O: 50.43
mips: 403.8, ram: 249.73, bw: 120.83, disk I/O: 40.38
mips: 593.02, ram: 494.63, bw: 442.42, disk I/O: 59.08
mips: 499.36, ram: 998.93, bw: 150.19, disk I/O: 49.92
sum mips: 34773.11, ram: 37471.14, bw: 14070.38, disk I/O: 3479.14
Percentages:
mips: 0.0172, ram: 0.0133, bw: 0.0322, disk I/O: 0.0172, time: 0.0153
mips: 0.0144, ram: 0.0266, bw: 0.0107, disk I/O: 0.0144, time: 0.0148
mips: 0.0174, ram: 0.0133, bw: 0.0086, disk I/O: 0.0175, time: 0.0153
mips: 0.0116, ram: 0.0066, bw: 0.0086, disk I/O: 0.0116, time: 0.0153
mips: 0.0175, ram: 0.0133, bw: 0.0324, disk I/O: 0.0174, time: 0.0153
mips: 0.0143, ram: 0.0267, bw: 0.0106, disk I/O: 0.0143, time: 0.0147
mips: 0.0175, ram: 0.0132, bw: 0.0087, disk I/O: 0.0174, time: 0.0153
mips: 0.0115, ram: 0.0067, bw: 0.0085, disk I/O: 0.0115, time: 0.0153
mips: 0.0173, ram: 0.0133, bw: 0.032, disk I/O: 0.0173, time: 0.0153
mips: 0.0143, ram: 0.0267, bw: 0.0107, disk I/O: 0.0143, time: 0.0147
mips: 0.0171, ram: 0.0133, bw: 0.0085, disk I/O: 0.0171, time: 0.0153
mips: 0.0116, ram: 0.0067, bw: 0.0086, disk I/O: 0.0116, time: 0.0153
mips: 0.0172, ram: 0.0135, bw: 0.0086, disk I/O: 0.0174, time: 0.0153
mips: 0.0174, ram: 0.0134, bw: 0.0321, disk I/O: 0.0173, time: 0.0153
mips: 0.0145, ram: 0.0266, bw: 0.0107, disk I/O: 0.0143, time: 0.0149
mips: 0.0175, ram: 0.0132, bw: 0.0086, disk I/O: 0.0175, time: 0.0153
mips: 0.0114, ram: 0.0066, bw: 0.0084, disk I/O: 0.0114, time: 0.0153
mips: 0.0174, ram: 0.0133, bw: 0.0322, disk I/O: 0.0173, time: 0.0153
mips: 0.0144, ram: 0.0267, bw: 0.0107, disk I/O: 0.0145, time: 0.0148
mips: 0.0171, ram: 0.0134, bw: 0.0084, disk I/O: 0.0172, time: 0.0153
mips: 0.0115, ram: 0.0065, bw: 0.0085, disk I/O: 0.0114, time: 0.0153
mips: 0.0175, ram: 0.0133, bw: 0.0325, disk I/O: 0.0175, time: 0.0153
mips: 0.0143, ram: 0.0267, bw: 0.0107, disk I/O: 0.0144, time: 0.0147
mips: 0.0116, ram: 0.0067, bw: 0.0085, disk I/O: 0.0115, time: 0.0153
mips: 0.0174, ram: 0.0133, bw: 0.0086, disk I/O: 0.0175, time: 0.0153
mips: 0.0115, ram: 0.0067, bw: 0.0086, disk I/O: 0.0116, time: 0.0153
mips: 0.0174, ram: 0.0135, bw: 0.0324, disk I/O: 0.0175, time: 0.0153
mips: 0.0145, ram: 0.0266, bw: 0.0107, disk I/O: 0.0144, time: 0.015
mips: 0.0172, ram: 0.0133, bw: 0.0085, disk I/O: 0.0173, time: 0.0153
mips: 0.0116, ram: 0.0067, bw: 0.0085, disk I/O: 0.0116, time: 0.0153
mips: 0.0171, ram: 0.0135, bw: 0.0317, disk I/O: 0.0172, time: 0.0153
mips: 0.0144, ram: 0.0265, bw: 0.0107, disk I/O: 0.0145, time: 0.0149
mips: 0.0171, ram: 0.0133, bw: 0.0084, disk I/O: 0.0171, time: 0.0153
mips: 0.0115, ram: 0.0066, bw: 0.0085, disk I/O: 0.0116, time: 0.0153
mips: 0.0174, ram: 0.0134, bw: 0.0323, disk I/O: 0.0174, time: 0.0153
mips: 0.0172, ram: 0.0133, bw: 0.0322, disk I/O: 0.0173, time: 0.0153
mips: 0.0144, ram: 0.0266, bw: 0.0107, disk I/O: 0.0144, time: 0.0149
```

```
mips: 0.0173, ram: 0.0134, bw: 0.0086, disk I/O: 0.0173, time: 0.0153
mips: 0.0115, ram: 0.0067, bw: 0.0085, disk I/O: 0.0114, time: 0.0153
mips: 0.017, ram: 0.0133, bw: 0.0315, disk I/O: 0.0171, time: 0.0153
mips: 0.0145, ram: 0.0268, bw: 0.0107, disk I/O: 0.0144, time: 0.0149
mips: 0.0176, ram: 0.0133, bw: 0.0086, disk I/O: 0.0176, time: 0.0153
mips: 0.0116, ram: 0.0068, bw: 0.0086, disk I/O: 0.0116, time: 0.0153
mips: 0.0175, ram: 0.0133, bw: 0.0324, disk I/O: 0.0175, time: 0.0153
mips: 0.0146, ram: 0.0266, bw: 0.0108, disk I/O: 0.0145, time: 0.015
mips: 0.0143, ram: 0.0266, bw: 0.0106, disk I/O: 0.0143, time: 0.0147
mips: 0.0172, ram: 0.0133, bw: 0.0085, disk I/O: 0.0172, time: 0.0153
mips: 0.0116, ram: 0.0065, bw: 0.0086, disk I/O: 0.0117, time: 0.0153
mips: 0.0173, ram: 0.0134, bw: 0.0319, disk I/O: 0.0173, time: 0.0153
mips: 0.0142, ram: 0.0266, bw: 0.0105, disk I/O: 0.0143, time: 0.0145
mips: 0.0174, ram: 0.0133, bw: 0.0086, disk I/O: 0.0173, time: 0.0153
mips: 0.0116, ram: 0.0067, bw: 0.0086, disk I/O: 0.0115, time: 0.0153
mips: 0.0172, ram: 0.0135, bw: 0.0317, disk I/O: 0.0171, time: 0.0153
mips: 0.0145, ram: 0.0267, bw: 0.0107, disk I/O: 0.0144, time: 0.0149
mips: 0.0171, ram: 0.0134, bw: 0.0085, disk I/O: 0.0172, time: 0.0153
mips: 0.0115, ram: 0.0068, bw: 0.0085, disk I/O: 0.0115, time: 0.0153
mips: 0.0171, ram: 0.0134, bw: 0.0084, disk I/O: 0.017, time: 0.0153
mips: 0.0172, ram: 0.0133, bw: 0.032, disk I/O: 0.0172, time: 0.0153
mips: 0.0144, ram: 0.0268, bw: 0.0107, disk I/O: 0.0144, time: 0.0149
mips: 0.0172, ram: 0.0133, bw: 0.0085, disk I/O: 0.0173, time: 0.0153
mips: 0.0113, ram: 0.0066, bw: 0.0084, disk I/O: 0.0113, time: 0.0153
mips: 0.0175, ram: 0.0133, bw: 0.0323, disk I/O: 0.0174, time: 0.0153
mips: 0.0145, ram: 0.0267, bw: 0.0108, disk I/O: 0.0145, time: 0.0149
mips: 0.0116, ram: 0.0067, bw: 0.0086, disk I/O: 0.0116, time: 0.0153
mips: 0.0171, ram: 0.0132, bw: 0.0314, disk I/O: 0.017, time: 0.0153
mips: 0.0144, ram: 0.0267, bw: 0.0107, disk I/O: 0.0143, time: 0.0148
Offsets:
mips: 0.0028, ram: 0.0067, bw: 0.0122, disk I/O: 0.0028
mips: 0.0044, ram: 0.0166, bw: 7.0E-4, disk I/O: 0.0044
mips: 0.0026, ram: 0.0067, bw: 0.0114, disk I/O: 0.0025
mips: 0.0084, ram: 0.0134, bw: 0.0114, disk I/O: 0.0084
mips: 0.0025, ram: 0.0067, bw: 0.0124, disk I/O: 0.0026
mips: 0.0043, ram: 0.0167, bw: 6.0E-4, disk I/O: 0.0043
mips: 0.0025, ram: 0.0068, bw: 0.0113, disk I/O: 0.0026
mips: 0.0085, ram: 0.0133, bw: 0.0115, disk I/O: 0.0085
mips: 0.0027, ram: 0.0067, bw: 0.012, disk I/O: 0.0027
mips: 0.0043, ram: 0.0167, bw: 7.0E-4, disk I/O: 0.0043
mips: 0.0029, ram: 0.0067, bw: 0.0115, disk I/O: 0.0029
mips: 0.0084, ram: 0.0133, bw: 0.0114, disk I/O: 0.0084
mips: 0.0028, ram: 0.0065, bw: 0.0114, disk I/O: 0.0026
mips: 0.0026, ram: 0.0066, bw: 0.0121, disk I/O: 0.0027
mips: 0.0045, ram: 0.0166, bw: 7.0E-4, disk I/O: 0.0043
mips: 0.0025, ram: 0.0068, bw: 0.0114, disk I/O: 0.0025
mips: 0.0086, ram: 0.0134, bw: 0.0116, disk I/O: 0.0086
mips: 0.0026, ram: 0.0067, bw: 0.0122, disk I/O: 0.0027
```

```
mips: 0.0044, ram: 0.0167, bw: 7.0E-4, disk I/O: 0.0045
mips: 0.0029, ram: 0.0066, bw: 0.0116, disk I/O: 0.0028
mips: 0.0085, ram: 0.0135, bw: 0.0115, disk I/O: 0.0086
mips: 0.0025, ram: 0.0067, bw: 0.0125, disk I/O: 0.0025
mips: 0.0043, ram: 0.0167, bw: 7.0E-4, disk I/O: 0.0044
mips: 0.0084, ram: 0.0133, bw: 0.0115, disk I/O: 0.0085
mips: 0.0026, ram: 0.0067, bw: 0.0114, disk I/O: 0.0025
mips: 0.0085, ram: 0.0133, bw: 0.0114, disk I/O: 0.0084
mips: 0.0026, ram: 0.0065, bw: 0.0124, disk I/O: 0.0025
mips: 0.0045, ram: 0.0166, bw: 7.0E-4, disk I/O: 0.0044
mips: 0.0028, ram: 0.0067, bw: 0.0115, disk I/O: 0.0027
mips: 0.0084, ram: 0.0133, bw: 0.0115, disk I/O: 0.0084
mips: 0.0029, ram: 0.0065, bw: 0.0117, disk I/O: 0.0028
mips: 0.0044, ram: 0.0165, bw: 7.0E-4, disk I/O: 0.0045
mips: 0.0029, ram: 0.0067, bw: 0.0116, disk I/O: 0.0029
mips: 0.0085, ram: 0.0134, bw: 0.0115, disk I/O: 0.0084
mips: 0.0026, ram: 0.0066, bw: 0.0123, disk I/O: 0.0026
mips: 0.0028, ram: 0.0067, bw: 0.0122, disk I/O: 0.0027
mips: 0.0044, ram: 0.0166, bw: 7.0E-4, disk I/O: 0.0044
mips: 0.0027, ram: 0.0066, bw: 0.0114, disk I/O: 0.0027
mips: 0.0085, ram: 0.0133, bw: 0.0115, disk I/O: 0.0086
mips: 0.003, ram: 0.0067, bw: 0.0115, disk I/O: 0.0029
mips: 0.0045, ram: 0.0168, bw: 7.0E-4, disk I/O: 0.0044
mips: 0.0024, ram: 0.0067, bw: 0.0114, disk I/O: 0.0024
mips: 0.0084, ram: 0.0132, bw: 0.0114, disk I/O: 0.0084
mips: 0.0025, ram: 0.0067, bw: 0.0124, disk I/O: 0.0025
mips: 0.0054, ram: 0.0066, bw: 0.0092, disk I/O: 0.0055
mips: 0.0043, ram: 0.0166, bw: 6.0E-4, disk I/O: 0.0043
mips: 0.0028, ram: 0.0067, bw: 0.0115, disk I/O: 0.0028
mips: 0.0084, ram: 0.0135, bw: 0.0114, disk I/O: 0.0083
mips: 0.0027, ram: 0.0066, bw: 0.0119, disk I/O: 0.0027
mips: 0.0042, ram: 0.0166, bw: 5.0E-4, disk I/O: 0.0043
mips: 0.0026, ram: 0.0067, bw: 0.0114, disk I/O: 0.0027
mips: 0.0084, ram: 0.0133, bw: 0.0114, disk I/O: 0.0085
mips: 0.0028, ram: 0.0065, bw: 0.0117, disk I/O: 0.0029
mips: 0.0045, ram: 0.0167, bw: 7.0E-4, disk I/O: 0.0044
mips: 0.0029, ram: 0.0066, bw: 0.0115, disk I/O: 0.0028
mips: 0.0085, ram: 0.0132, bw: 0.0115, disk I/O: 0.0085
mips: 0.0029, ram: 0.0066, bw: 0.0116, disk I/O: 0.003
mips: 0.0028, ram: 0.0067, bw: 0.012, disk I/O: 0.0028
mips: 0.0044, ram: 0.0168, bw: 7.0E-4, disk I/O: 0.0044
mips: 0.0028, ram: 0.0067, bw: 0.0115, disk I/O: 0.0027
mips: 0.0087, ram: 0.0134, bw: 0.0116, disk I/O: 0.0087
mips: 0.0025, ram: 0.0067, bw: 0.0123, disk I/O: 0.0026
mips: 0.0045, ram: 0.0167, bw: 8.0E-4, disk I/O: 0.0045
mips: 0.0084, ram: 0.0133, bw: 0.0114, disk I/O: 0.0084
mips: 0.0029, ram: 0.0068, bw: 0.0114, disk I/O: 0.003
mips: 0.0044, ram: 0.0167, bw: 7.0E-4, disk I/O: 0.0043
```

```
sum mips: 0.3006, ram: 0.7064, bw: 0.5919, disk I/O: 0.3004
```

```
0.02: Host #0
totalRequestedCPU: 597.9657468407776, mipsShare: [400.0]
totalRequestedBandwidth: 456.2622067874816, bwShare: 1000.0
totalRequestedStorageIO: 62.434219507227795, storageShare: 936.6925932221369
cpuDampingFactor: 1.494914367101944
Requested CPU: 597.9657468407776, setUtilization: 400.0
cloudlet finishedSoFar: 0, utilizationOfRam: 500.0; getRemainingCloudletLength() 58464000000
nextChangeTime: 0.01
totalRequestedCPU: 391.2752921940316, mipsShare: [307.44632531076553]
totalRequestedBandwidth: 123.77268406855157, bwShare: 1000.0
totalRequestedStorageIO: 38.091074773588396, storageShare: 571.4755127860664
cpuDampingFactor: 1.2726621201230233
Requested CPU: 391.2752921940316, setUtilization: 307.4463253107655
cloudlet finishedSoFar: 0, utilizationOfRam: 995.0; getRemainingCloudletLength() 39899000000
nextChangeTime: 0.01
totalRequestedCPU: 571.2276588977794, mipsShare: [448.84470894958446]
totalRequestedBandwidth: 118.81976783695025, bwShare: 1000.0
totalRequestedStorageIO: 57.88621102077325, storageShare: 868.459404807245
cpuDampingFactor: 1.272662120123023
Requested CPU: 571.2276588977794, setUtilization: 448.8447089495845
cloudlet finishedSoFar: 0, utilizationOfRam: 510.0; getRemainingCloudletLength() 59978000000
nextChangeTime: 0.01
totalRequestedCPU: 391.51201489709763, mipsShare: [314.3018038136733]
totalRequestedBandwidth: 122.23299021586384, bwShare: 1000.0
totalRequestedStorageIO: 43.07486674311885, storageShare: 646.2467049442179
cpuDampingFactor: 1.2456562773314424
Requested CPU: 391.51201489709763, setUtilization: 314.3018038136733
cloudlet finishedSoFar: 0, utilizationOfRam: 263.0; getRemainingCloudletLength() 40140000000
nextChangeTime: 0.01
totalRequestedCPU: 584.8555423134582, mipsShare: [400.0]
totalRequestedBandwidth: 471.54632796460317, bwShare: 1000.0
totalRequestedStorageIO: 65.12924959679435, storageShare: 977.1257842403339
cpuDampingFactor: 1.4621388557836454
Requested CPU: 584.8555423134582, setUtilization: 400.0
cloudlet finishedSoFar: 0, utilizationOfRam: 494.0; getRemainingCloudletLength() 59922000000
nextChangeTime: 0.01
0.020000: [Host #0] Total allocated MIPS for VM #0 (Host #0) is 400.00, was requested 597.97
0.020000: [Host #0] Under allocated MIPS for VM #0: 197.97
0.020000: [Host #0] Total allocated MIPS for VM #1 (Host #0) is 307.45, was requested 391.28
0.020000: [Host #0] Under allocated MIPS for VM #1: 83.83
0.020000: [Host #0] Total allocated MIPS for VM #2 (Host #0) is 448.84, was requested 571.23
0.020000: [Host #0] Under allocated MIPS for VM #2: 122.38
0.020000: [Host #0] Total allocated MIPS for VM #3 (Host #0) is 314.30, was requested 391.51
0.020000: [Host #0] Under allocated MIPS for VM #3: 77.21
0.020000: [Host #0] Total allocated MIPS for VM #4 (Host #0) is 400.00, was requested 584.86
0.020000: [Host #0] Under allocated MIPS for VM #4: 184.86
```

Appendix B

FFMRA traces

CLID	STATUS	VM ID	ST	FT	Users
132	SUCCESS 2	132	0.01	6.78	user_0
135	SUCCESS 2	135	0.01	6.78	user_0
138	SUCCESS 2	138	0.01	6.78	user_0
141	SUCCESS 2	141	0.01	6.78	user_0
144	SUCCESS 2	144	0.01	6.78	user_0
147	SUCCESS 2	147	0.01	6.78	user_0
133	SUCCESS 2	133	0.01	7.09	user_1
134	SUCCESS 2	134	0.01	7.09	user_2
136	SUCCESS 2	136	0.01	7.09	user_1
137	SUCCESS 2	137	0.01	7.09	user_2
139	SUCCESS 2	139	0.01	7.09	user_1
140	SUCCESS 2	140	0.01	7.09	user_2
142	SUCCESS 2	142	0.01	7.09	user_1
143	SUCCESS 2	143	0.01	7.09	user_2
145	SUCCESS 2	145	0.01	7.09	user_1
146	SUCCESS 2	146	0.01	7.09	user_2
148	SUCCESS 2	148	0.01	7.09	user_1
149	SUCCESS 2	149	0.01	7.09	user_2
66	SUCCESS 2	66	0.01	10.9	user_0
69	SUCCESS 2	69	0.01	10.9	user_0
72	SUCCESS 2	72	0.01	10.9	user_0
75	SUCCESS 2	75	0.01	10.9	user_0
78	SUCCESS 2	78	0.01	10.9	user_0
81	SUCCESS 2	81	0.01	10.9	user_0
84	SUCCESS 2	84	0.01	10.9	user_0
87	SUCCESS 2	87	0.01	10.9	user_0
90	SUCCESS 2	90	0.01	10.9	user_0
93	SUCCESS 2	93	0.01	10.9	user_0
96	SUCCESS 2	96	0.01	10.9	user_0
0	SUCCESS 2	0	0.01	11	user_0
3	SUCCESS 2	3	0.01	11	user_0

```
6       SUCCESS 2       6       0.01    11      user_0
9       SUCCESS 2       9       0.01    11      user_0
12      SUCCESS 2       12      0.01    11      user_0
15      SUCCESS 2       15      0.01    11      user_0
18      SUCCESS 2       18      0.01    11      user_0
21      SUCCESS 2       21      0.01    11      user_0
24      SUCCESS 2       24      0.01    11      user_0
27      SUCCESS 2       27      0.01    11      user_0
30      SUCCESS 2       30      0.01    11.01   user_0
36      SUCCESS 2       36      0.01    11.07   user_0
39      SUCCESS 2       39      0.01    11.07   user_0
51      SUCCESS 2       51      0.01    11.07   user_0
57      SUCCESS 2       57      0.01    11.07   user_0
60      SUCCESS 2       60      0.01    11.07   user_0
33      SUCCESS 2       33      0.01    11.08   user_0
42      SUCCESS 2       42      0.01    11.08   user_0
45      SUCCESS 2       45      0.01    11.08   user_0
48      SUCCESS 2       48      0.01    11.08   user_0
54      SUCCESS 2       54      0.01    11.08   user_0
63      SUCCESS 2       63      0.01    11.08   user_0
99      SUCCESS 2       99      0.01    11.48   user_0
102     SUCCESS 2       102     0.01    11.48   user_0
105     SUCCESS 2       105     0.01    11.48   user_0
108     SUCCESS 2       108     0.01    11.48   user_0
111     SUCCESS 2       111     0.01    11.48   user_0
114     SUCCESS 2       114     0.01    11.48   user_0
117     SUCCESS 2       117     0.01    11.48   user_0
120     SUCCESS 2       120     0.01    11.48   user_0
123     SUCCESS 2       123     0.01    11.48   user_0
126     SUCCESS 2       126     0.01    11.48   user_0
129     SUCCESS 2       129     0.01    11.48   user_0
1       SUCCESS 2       1       0.01    11.96   user_1
2       SUCCESS 2       2       0.01    11.96   user_2
4       SUCCESS 2       4       0.01    11.96   user_1
5       SUCCESS 2       5       0.01    11.96   user_2
7       SUCCESS 2       7       0.01    11.96   user_1
8       SUCCESS 2       8       0.01    11.96   user_2
10      SUCCESS 2       10      0.01    11.96   user_1
11      SUCCESS 2       11      0.01    11.96   user_2
13      SUCCESS 2       13      0.01    11.96   user_1
14      SUCCESS 2       14      0.01    11.96   user_2
16      SUCCESS 2       16      0.01    11.96   user_1
17      SUCCESS 2       17      0.01    11.96   user_2
19      SUCCESS 2       19      0.01    11.96   user_1
20      SUCCESS 2       20      0.01    11.96   user_2
22      SUCCESS 2       22      0.01    11.96   user_1
23      SUCCESS 2       23      0.01    11.96   user_2
25      SUCCESS 2       25      0.01    11.96   user_1
```

```
26      SUCCESS 2       26      0.01    11.96   user_2
28      SUCCESS 2       28      0.01    11.96   user_1
29      SUCCESS 2       29      0.01    11.96   user_2
31      SUCCESS 2       31      0.01    11.96   user_1
32      SUCCESS 2       32      0.01    11.96   user_2
34      SUCCESS 2       34      0.01    12.07   user_1
35      SUCCESS 2       35      0.01    12.07   user_2
37      SUCCESS 2       37      0.01    12.07   user_1
38      SUCCESS 2       38      0.01    12.07   user_2
40      SUCCESS 2       40      0.01    12.07   user_1
41      SUCCESS 2       41      0.01    12.07   user_2
43      SUCCESS 2       43      0.01    12.07   user_1
44      SUCCESS 2       44      0.01    12.07   user_2
46      SUCCESS 2       46      0.01    12.07   user_1
47      SUCCESS 2       47      0.01    12.07   user_2
49      SUCCESS 2       49      0.01    12.07   user_1
50      SUCCESS 2       50      0.01    12.07   user_2
52      SUCCESS 2       52      0.01    12.07   user_1
53      SUCCESS 2       53      0.01    12.07   user_2
55      SUCCESS 2       55      0.01    12.07   user_1
56      SUCCESS 2       56      0.01    12.07   user_2
58      SUCCESS 2       58      0.01    12.07   user_1
59      SUCCESS 2       59      0.01    12.07   user_2
61      SUCCESS 2       61      0.01    12.07   user_1
62      SUCCESS 2       62      0.01    12.07   user_2
64      SUCCESS 2       64      0.01    12.07   user_1
65      SUCCESS 2       65      0.01    12.07   user_2
67      SUCCESS 2       67      0.01    12.1    user_1
68      SUCCESS 2       68      0.01    12.1    user_2
70      SUCCESS 2       70      0.01    12.1    user_1
71      SUCCESS 2       71      0.01    12.1    user_2
73      SUCCESS 2       73      0.01    12.1    user_1
74      SUCCESS 2       74      0.01    12.1    user_2
76      SUCCESS 2       76      0.01    12.1    user_1
77      SUCCESS 2       77      0.01    12.1    user_2
79      SUCCESS 2       79      0.01    12.1    user_1
80      SUCCESS 2       80      0.01    12.1    user_2
82      SUCCESS 2       82      0.01    12.1    user_1
83      SUCCESS 2       83      0.01    12.1    user_2
85      SUCCESS 2       85      0.01    12.1    user_1
86      SUCCESS 2       86      0.01    12.1    user_2
88      SUCCESS 2       88      0.01    12.1    user_1
89      SUCCESS 2       89      0.01    12.1    user_2
91      SUCCESS 2       91      0.01    12.1    user_1
92      SUCCESS 2       92      0.01    12.1    user_2
94      SUCCESS 2       94      0.01    12.1    user_1
95      SUCCESS 2       95      0.01    12.1    user_2
97      SUCCESS 2       97      0.01    12.1    user_1
```

```
98      SUCCESS 2       98      0.01    12.1    user_2
101     SUCCESS 2       101     0.01    12.46   user_2
113     SUCCESS 2       113     0.01    12.46   user_2
128     SUCCESS 2       128     0.01    12.46   user_2
100     SUCCESS 2       100     0.01    12.47   user_1
103     SUCCESS 2       103     0.01    12.47   user_1
104     SUCCESS 2       104     0.01    12.47   user_2
106     SUCCESS 2       106     0.01    12.47   user_1
107     SUCCESS 2       107     0.01    12.47   user_2
109     SUCCESS 2       109     0.01    12.47   user_1
110     SUCCESS 2       110     0.01    12.47   user_2
112     SUCCESS 2       112     0.01    12.47   user_1
115     SUCCESS 2       115     0.01    12.47   user_1
116     SUCCESS 2       116     0.01    12.47   user_2
118     SUCCESS 2       118     0.01    12.47   user_1
119     SUCCESS 2       119     0.01    12.47   user_2
121     SUCCESS 2       121     0.01    12.47   user_1
122     SUCCESS 2       122     0.01    12.47   user_2
124     SUCCESS 2       124     0.01    12.47   user_1
125     SUCCESS 2       125     0.01    12.47   user_2
127     SUCCESS 2       127     0.01    12.47   user_1
130     SUCCESS 2       130     0.01    12.47   user_1
131     SUCCESS 2       131     0.01    12.47   user_2

Resources:
mips: 14015.83, ram: 43.97, bw: 1403.67, disk I/O: 139598.17
mips: 29016.17, ram: 0.0, bw: 2932.17, disk I/O: 37465.83
mips: 23176.67, ram: 0.0, bw: 7599.17, disk I/O: 37623.33
sum mips: 66208.67, ram: 43.97, bw: 11935.0, disk I/O: 214687.33
Percentages:
mips: 0.2117, ram: 1.0, bw: 0.1176, disk I/O: 0.6502, time: 0.3145
mips: 0.4383, ram: 0.0, bw: 0.2457, disk I/O: 0.1745, time: 0.3427
mips: 0.3501, ram: 0.0, bw: 0.6367, disk I/O: 0.1752, time: 0.3427
Offsets:
mips: 0.0983, ram: 0.69, bw: 0.1924, disk I/O: 0.3402
mips: 0.0983, ram: 0.34, bw: 0.0943, disk I/O: 0.1655
mips: 0.0101, ram: 0.34, bw: 0.2967, disk I/O: 0.1648
sum mips: 0.2066, ram: 1.37, bw: 0.5834, disk I/O: 0.6705
```

Appendix C

HFFMRA traces

CL ID	STATUS	VM ID	ST	FT	Users
0	SUCCESS	0	0.01	12.26	user_0
63	SUCCESS	63	0.01	12.33	user_3
84	SUCCESS	84	0.01	12.37	user_0
12	SUCCESS	12	0.01	12.39	user_0
36	SUCCESS	36	0.01	12.4	user_0
57	SUCCESS	57	0.01	12.41	user_1
66	SUCCESS	66	0.01	12.43	user_2
81	SUCCESS	81	0.01	12.51	user_1
48	SUCCESS	48	0.01	12.54	user_0
75	SUCCESS	75	0.01	12.77	user_3
18	SUCCESS	18	0.01	13.06	user_2
54	SUCCESS	54	0.01	13.16	user_2
87	SUCCESS	87	0.01	13.22	user_3
15	SUCCESS	15	0.01	13.24	user_3
42	SUCCESS	42	0.01	13.41	user_2
60	SUCCESS	60	0.01	13.5	user_0
69	SUCCESS	69	0.01	13.52	user_1
39	SUCCESS	39	0.01	13.62	user_3
21	SUCCESS	21	0.01	13.64	user_1
30	SUCCESS	30	0.01	13.71	user_2
45	SUCCESS	45	0.01	13.72	user_1
51	SUCCESS	51	0.01	13.91	user_3
24	SUCCESS	24	0.01	13.94	user_0
78	SUCCESS	78	0.01	13.98	user_2
9	SUCCESS	9	0.01	14.01	user_1
33	SUCCESS	33	0.01	14.18	user_1
27	SUCCESS	27	0.01	14.3	user_3
6	SUCCESS	6	0.01	14.42	user_2
72	SUCCESS	72	0.01	15.01	user_0
67	SUCCESS	67	0.01	15.93	user_3
68	SUCCESS	68	0.01	15.93	user_0

```
49      SUCCESS 49      0.01    15.96   user_1
50      SUCCESS 50      0.01    15.96   user_2
76      SUCCESS 76      0.01    15.97   user_0
77      SUCCESS 77      0.01    15.98   user_1
13      SUCCESS 13      0.01    16.36   user_1
14      SUCCESS 14      0.01    16.36   user_2
43      SUCCESS 43      0.01    16.39   user_3
44      SUCCESS 44      0.01    16.39   user_0
22      SUCCESS 22      0.01    16.49   user_2
23      SUCCESS 23      0.01    16.49   user_3
79      SUCCESS 79      0.01    16.53   user_3
80      SUCCESS 80      0.01    16.53   user_0
3       SUCCESS 3       0.01    16.57   user_3
82      SUCCESS 82      0.01    16.62   user_2
83      SUCCESS 83      0.01    16.62   user_3
58      SUCCESS 58      0.01    16.75   user_2
59      SUCCESS 59      0.01    16.75   user_3
25      SUCCESS 25      0.01    16.85   user_1
26      SUCCESS 26      0.01    16.85   user_2
70      SUCCESS 70      0.01    16.91   user_2
71      SUCCESS 71      0.01    16.91   user_3
34      SUCCESS 34      0.01    16.96   user_2
35      SUCCESS 35      0.01    16.96   user_3
55      SUCCESS 55      0.01    16.98   user_3
56      SUCCESS 56      0.01    16.98   user_0
16      SUCCESS 16      0.01    17.03   user_0
17      SUCCESS 17      0.01    17.03   user_1
1       SUCCESS 1       0.01    17.07   user_1
2       SUCCESS 2       0.01    17.07   user_2
85      SUCCESS 85      0.01    17.08   user_1
86      SUCCESS 86      0.01    17.08   user_2
64      SUCCESS 64      0.01    17.12   user_0
65      SUCCESS 65      0.01    17.12   user_1
52      SUCCESS 52      0.01    17.13   user_0
53      SUCCESS 53      0.01    17.13   user_1
19      SUCCESS 19      0.01    17.17   user_3
20      SUCCESS 20      0.01    17.17   user_0
37      SUCCESS 37      0.01    17.31   user_1
38      SUCCESS 38      0.01    17.31   user_2
7       SUCCESS 7       0.01    17.37   user_3
8       SUCCESS 8       0.01    17.37   user_0
61      SUCCESS 61      0.01    17.41   user_1
62      SUCCESS 62      0.01    17.41   user_2
32      SUCCESS 32      0.01    17.53   user_0
40      SUCCESS 40      0.01    17.53   user_0
41      SUCCESS 41      0.01    17.53   user_1
31      SUCCESS 31      0.01    17.54   user_3
10      SUCCESS 10      0.01    18      user_2
```

```
11      SUCCESS 11      0.01    18.01   user_3
73      SUCCESS 73      0.01    18.06   user_1
74      SUCCESS 74      0.01    18.06   user_2
88      SUCCESS 88      0.01    18.24   user_0
89      SUCCESS 89      0.01    18.24   user_1
4       SUCCESS 4       0.01    18.62   user_0
5       SUCCESS 5       0.01    18.62   user_1
28      SUCCESS 28      0.01    18.66   user_0
29      SUCCESS 29      0.01    18.66   user_1
46      SUCCESS 46      0.01    19.78   user_2
47      SUCCESS 47      0.01    19.78   user_3

------------------------------------------------

0.07: Host #14
totalRequestedCPU: 226.7586679442413, mipsShare: [379.1484193599119]
totalRequestedBandwidth: 23.040222009293117, bwShare: 23.040222009293117
totalRequestedStorageIO: 1643.6435521486899, storageShare: 1241.7031612801763
storageDampingFactor: 1.3237008678097584
Requested CPU: 226.7586679442413, setUtilization: 171.3065794989196
cloudlet finishedSoFar: 8570943, utilizationOfRam: 0.0; getRemainingCloudletLength() 2292429057
nextChangeTime: 0.01
totalRequestedCPU: 514.3907942086543, mipsShare: [343.5149749058453]
totalRequestedBandwidth: 66.41380868701862, bwShare: 66.41380868701862
totalRequestedStorageIO: 666.0976985521636, storageShare: 375.60895886439755
cpuDampingFactor: 1.4974333923860077
storageDampingFactor: 1.7733807536593886
Requested CPU: 514.3907942086543, setUtilization: 290.0622402420934
cloudlet finishedSoFar: 14502681, utilizationOfRam: 0.0; getRemainingCloudletLength() 5249497319
nextChangeTime: 0.01
totalRequestedCPU: 415.2930943574481, mipsShare: [277.33660573424294]
totalRequestedBandwidth: 134.3256603971646, bwShare: 134.3256603971646
totalRequestedStorageIO: 678.6513207943293, storageShare: 382.6878798554263
cpuDampingFactor: 1.4974333923860077
storageDampingFactor: 1.7733807536593884
Requested CPU: 415.2930943574481, setUtilization: 234.1815729647944
cloudlet finishedSoFar: 11709012, utilizationOfRam: 0.0; getRemainingCloudletLength() 4144290988
nextChangeTime: 0.01
0.070000: [Host #14] Total allocated MIPS for VM #42 (Host #14) is 379.15, was requested 226.77
0.070000: [Host #14] Total allocated MIPS for VM #43 (Host #14) is 343.51, was requested 514.34
0.070000: [Host #14] Under allocated MIPS for VM #43: 170.82
0.070000: [Host #14] Total allocated MIPS for VM #44 (Host #14) is 277.34, was requested 415.25
0.070000: [Host #14] Under allocated MIPS for VM #44: 137.92
```

Appendix D

MRFS traces

This is note that due to the policies under MRFs, some VMs are not scheduled in some hosts to make sure that the tasks are assigned to the most efficient server and also a balance in the number of dominant resources is maintained. Accordingly, some host assignments for a specific number of VMs are failed.

```
- DatacenterSimple1 is starting...
- DatacenterBrokerSimple2 is starting...
- Entities started.
- 0.00: DatacenterBrokerSimple2: List of 1 datacenters(s) received.
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 0 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 1 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 2 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 3 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 4 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 5 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 6 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 7 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 8 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 9 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 10 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 11 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 12 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 13 in DatacenterSimple1
- 0.00: DatacenterBrokerSimple2: Trying to create Vm 14 in DatacenterSimple1
- 0.00: VmAllocationPolicyMrfs: Vm 0 has been allocated to Host 0/DC 1
- 0.00: VmAllocationPolicyMrfs: Vm 1 has been allocated to Host 0/DC 1
- 0.00: VmAllocationPolicyMrfs: Vm 2 has been allocated to Host 1/DC 1
- 0.00: VmAllocationPolicyMrfs: Vm 3 has been allocated to Host 1/DC 1
- 0.00: VmAllocationPolicyMrfs: Vm 4 has been allocated to Host 1/DC 1
```

```
- 0.00: VmAllocationPolicyMrfs: Vm 5 has been allocated to Host 1/DC 1
- [1;31mERROR 0.00: Creation of Vm 6 on Host 1/DC 1 failed
- 0.00: VmAllocationPolicyMrfs: Vm 6 has been allocated to Host 3/DC 1
- [1;31mERROR 0.00: Creation of Vm 7 on Host 1/DC 1 failed
- 0.00: VmAllocationPolicyMrfs: Vm 7 has been allocated to Host 3/DC 1
- [1;31mERROR 0.00: Creation of Vm 8 on Host 1/DC 1 failed
- 0.00: VmAllocationPolicyMrfs: Vm 8 has been allocated to Host 3/DC 1
- [1;31mERROR 0.00: Creation of Vm 9 on Host 1/DC 1 failed
- 0.00: VmAllocationPolicyMrfs: Vm 9 has been allocated to Host 3/DC 1
- [1;31mERROR 0.00: Creation of Vm 10 on Host 1/DC 1 failed
- [1;31mERROR 0.00: Creation of Vm 10 on Host 3/DC 1 failed
- 0.00: VmAllocationPolicyMrfs: Vm 10 has been allocated to Host 2/DC 1
- [1;31mERROR 0.00: Creation of Vm 11 on Host 1/DC 1 failed
- [1;31mERROR 0.00: Creation of Vm 11 on Host 3/DC 1 failed
- 0.00: VmAllocationPolicyMrfs: Vm 11 has been allocated to Host 0/DC 1
- 0.00: VmAllocationPolicyMrfs: Vm 12 has been allocated to Host 2/DC 1
- 0.00: VmAllocationPolicyMrfs: Vm 13 has been allocated to Host 2/DC 1
- 0.00: VmAllocationPolicyMrfs: Vm 14 has been allocated to Host 2/DC 1
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 0 to Vm 0 in Host 0/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 1 to Vm 1 in Host 0/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 2 to Vm 2 in Host 1/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 3 to Vm 3 in Host 1/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 4 to Vm 4 in Host 1/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 5 to Vm 5 in Host 1/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 6 to Vm 6 in Host 3/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 7 to Vm 7 in Host 3/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 8 to Vm 8 in Host 3/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 9 to Vm 9 in Host 3/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 10 to Vm 10 in Host 2/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 11 to Vm 11 in Host 0/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 12 to Vm 12 in Host 2/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 13 to Vm 13 in Host 2/DC 1.
- 0.10: DatacenterBrokerSimple2: Sending Cloudlet 14 to Vm 14 in Host 2/DC 1.
- 0.10: DatacenterBrokerSimple2: All waiting Cloudlets submitted to some VM.
- 20.11: DatacenterBrokerSimple2: Cloudlet 0 finished in Vm 0 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 1 finished in Vm 1 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 11 finished in Vm 11 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 2 finished in Vm 2 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 3 finished in Vm 3 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 4 finished in Vm 4 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 5 finished in Vm 5 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 10 finished in Vm 10 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 12 finished in Vm 12 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 13 finished in Vm 13 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 14 finished in Vm 14 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 6 finished in Vm 6 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 7 finished in Vm 7 and returned to broker.
- 20.11: DatacenterBrokerSimple2: Cloudlet 8 finished in Vm 8 and returned to broker.
```

```
- 20.11: DatacenterBrokerSimple2: Cloudlet 9 finished in Vm 9 and returned to broker.
- 20.22: Processing last events before simulation shutdown.
- 20.22: DatacenterBrokerSimple2 is shutting down...
- 20.22: DatacenterBrokerSimple2: Requesting Vm 14 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 13 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 12 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 11 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 10 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 9 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 8 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 7 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 6 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 5 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 4 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 3 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 2 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 1 destruction.
- 20.22: DatacenterBrokerSimple2: Requesting Vm 0 destruction.
- 20.22: DatacenterSimple: Vm 14 destroyed on Host 2/DC 1.
- 20.22: DatacenterSimple: Vm 13 destroyed on Host 2/DC 1.
- 20.22: DatacenterSimple: Vm 12 destroyed on Host 2/DC 1.
- 20.22: DatacenterSimple: Vm 11 destroyed on Host 0/DC 1.
- 20.22: DatacenterSimple: Vm 10 destroyed on Host 2/DC 1.
- 20.22: DatacenterSimple: Vm 9 destroyed on Host 3/DC 1.
- 20.22: DatacenterSimple: Vm 8 destroyed on Host 3/DC 1.
- 20.22: DatacenterSimple: Vm 7 destroyed on Host 3/DC 1.
- 20.22: DatacenterSimple: Vm 6 destroyed on Host 3/DC 1.
- 20.22: DatacenterSimple: Vm 5 destroyed on Host 1/DC 1.
- 20.22: DatacenterSimple: Vm 4 destroyed on Host 1/DC 1.
- 20.22: DatacenterSimple: Vm 3 destroyed on Host 1/DC 1.
- 20.22: DatacenterSimple: Vm 2 destroyed on Host 1/DC 1.
- 20.22: DatacenterSimple: Vm 1 destroyed on Host 0/DC 1.
- 20.22: DatacenterSimple: Vm 0 destroyed on Host 0/DC 1.
- Simulation: No more future events

- CloudInformationService0: Notify all CloudSim Plus entities to shutdown.

- 20.22: DatacenterSimple1 is shutting down...
-

                                        SIMULATION RESULTS

Cloudlet|Status |DC|Host|Host PEs |VM|StartTime|FinishTime|ExecTime
      ID|       |ID|  ID|CPU cores|ID|Seconds  |   Seconds| Seconds

       0|SUCCESS| 1|   0|        8| 0|    0    |        20|       20
       1|SUCCESS| 1|   0|        8| 1|    0    |        20|       20
```

```
  2|SUCCESS| 1|   1|        8| 2|   0     |        20|      20
  3|SUCCESS| 1|   1|        8| 3|   0     |        20|      20
  4|SUCCESS| 1|   1|        8| 4|   0     |        20|      20
  5|SUCCESS| 1|   1|        8| 5|   0     |        20|      20
  6|SUCCESS| 1|   3|        8| 6|   0     |        20|      20
  7|SUCCESS| 1|   3|        8| 7|   0     |        20|      20
  8|SUCCESS| 1|   3|        8| 8|   0     |        20|      20
  9|SUCCESS| 1|   3|        8| 9|   0     |        20|      20
 10|SUCCESS| 1|   2|        8|10|   0     |        20|      20
 11|SUCCESS| 1|   0|        8|11|   0     |        20|      20
 12|SUCCESS| 1|   2|        8|12|   0     |        20|      20
 13|SUCCESS| 1|   2|        8|13|   0     |        20|      20
 14|SUCCESS| 1|   2|        8|14|   0     |        20|      20
```

Vms 0,1,2,3,4 RAM and BW utilization history

```
Time:        1.1 secs | RAM Utilization:       0.00% | BW Utilization:       0.00%
Time:        2.1 secs | RAM Utilization:      84.40% | BW Utilization:      44.40%
Time:        3.1 secs | RAM Utilization:      88.40% | BW Utilization:      48.40%
Time:        4.1 secs | RAM Utilization:      92.40% | BW Utilization:      52.40%
Time:        5.1 secs | RAM Utilization:      96.40% | BW Utilization:      56.40%
Time:        6.1 secs | RAM Utilization:     100.00% | BW Utilization:      60.40%
Time:        7.1 secs | RAM Utilization:     100.00% | BW Utilization:      64.40%
Time:        8.1 secs | RAM Utilization:     100.00% | BW Utilization:      68.40%
Time:        9.1 secs | RAM Utilization:     100.00% | BW Utilization:      72.40%
Time:       10.1 secs | RAM Utilization:     100.00% | BW Utilization:      76.40%
Time:       11.1 secs | RAM Utilization:     100.00% | BW Utilization:      80.40%
Time:       12.1 secs | RAM Utilization:     100.00% | BW Utilization:      84.40%
Time:       13.1 secs | RAM Utilization:     100.00% | BW Utilization:      88.40%
Time:       14.1 secs | RAM Utilization:     100.00% | BW Utilization:      92.40%
Time:       15.0 secs | RAM Utilization:     100.00% | BW Utilization:      96.40%
Time:       15.1 secs | RAM Utilization:     100.00% | BW Utilization:      99.60%
Time:       15.2 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       15.3 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       16.3 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       17.3 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       18.3 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       19.3 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       19.8 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       20.1 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       20.2 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
Time:       20.3 secs | RAM Utilization:     100.00% | BW Utilization:     100.00%
```

Vms 5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
20,21,22,23,24 and 25 RAM and BW utilization history

```
Time:        1.1 secs | RAM Utilization:       0.00% | BW Utilization:       0.00%
```

```
Time:        2.1 secs | RAM Utilization:      63.30% | BW Utilization:      33.30%
Time:        3.1 secs | RAM Utilization:      66.30% | BW Utilization:      36.30%
Time:        4.1 secs | RAM Utilization:      69.30% | BW Utilization:      39.30%
Time:        5.1 secs | RAM Utilization:      72.30% | BW Utilization:      42.30%
Time:        6.1 secs | RAM Utilization:      75.30% | BW Utilization:      45.30%
Time:        7.1 secs | RAM Utilization:      78.30% | BW Utilization:      48.30%
Time:        8.1 secs | RAM Utilization:      81.30% | BW Utilization:      51.30%
Time:        9.1 secs | RAM Utilization:      84.30% | BW Utilization:      54.30%
Time:       10.1 secs | RAM Utilization:      87.30% | BW Utilization:      57.30%
Time:       11.1 secs | RAM Utilization:      90.30% | BW Utilization:      60.30%
Time:       12.1 secs | RAM Utilization:      93.30% | BW Utilization:      63.30%
Time:       13.1 secs | RAM Utilization:      96.30% | BW Utilization:      66.30%
Time:       14.1 secs | RAM Utilization:      99.30% | BW Utilization:      69.30%
Time:       15.0 secs | RAM Utilization:     100.00% | BW Utilization:      72.30%
Time:       15.1 secs | RAM Utilization:     100.00% | BW Utilization:      74.70%
Time:       15.2 secs | RAM Utilization:     100.00% | BW Utilization:      75.30%
Time:       15.3 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       16.3 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       17.3 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       18.3 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       19.3 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       19.8 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       20.1 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       20.2 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
Time:       20.3 secs | RAM Utilization:     100.00% | BW Utilization:      75.60%
--------------------------------------------------------------------------------
```

LISTING D.1: The trace for bandwidth and RAM utilization for 25 VMS and 80 cloudlets under MRFS policy

Bibliography

Abdelzahir, A., J, D. N., Imran, G., Abubakar, E. and Kitchenham, B. (2015), 'Quality of service approaches in cloud computing: A systematic mapping study', *Journal of Systems and Software* pp. 159 – 179.

Ahmed, R., Qadeer, M. A. and Ahmad, M. (2009), Bandwidth resource allocation, *in* '2009 International Conference on Future Computer and Communication', pp. 344–348.

Allalouf, M. and Shavitt, Y. (2004), Maximum flow routing with weighted max-min fairness, *in* J. Solé-Pareta, M. Smirnov, P. Van Mieghem, J. Domingo-Pascual, E. Monteiro, P. Reichl, B. Stiller and R. J. Gibbens, eds, 'Quality of Service in the Emerging Networking Panorama', Springer Berlin Heidelberg, pp. 278–287.

Altman, E., Avrachenkov, K. and Garnaev, A. (2008), Generalized -fair resource allocation in wireless networks, *in* '2008 47th IEEE Conference on Decision and Control', pp. 2414–2419.

Ardagna, D. and Squillante, M. (2015), 'Special issue on performance and resource management in big data applications', *ACM SIGMETRICS Performance Evaluation Review* **42**, 2–2.

Avital, G. and Noam, N. (2012), 'Fair allocation without trade'.

Aziz, H., Chan, H. and Li, B. (2019), 'Weighted maxmin fair share allocation of indivisible chores', *CoRR* **abs/1906.07602**.

Babaioff, M., Nisan, N. and Talgam-Cohen, I. (2019), Fair allocation through competitive equilibrium from generic incomes, FAT* '19, Association for Computing Machinery, p. 180.

Baruah, S., Cohen, N., Plaxton, C. and Varvel, D. (1993), Proportionate progress: a notion of fairness in resource allocation., pp. 345–354.

Beltre, A., Saha, P. and Govindaraju, M. (2019), Kubesphere: An approach to multi-tenant fair scheduling for kubernetes clusters.

Bernstein, D. (2014), 'Containers and cloud: From lxc to docker to kubernetes', *IEEE Cloud Computing* **1**(3), 81–84.

Bertsimas, D., Farias, V. F. and Trichakis, N. (2012), 'On the efficiency-fairness trade-off', *Manage. Sci.* .

Bertsimas, D., Farias, V. and Trichakis, N. (2011), 'The price of fairness', *Operations Research* pp. 17–31.

Bhattacharya, A., Culler, D., Friedman, E., Ghodsi, A., Shenker, S. and Stoica, I. (2013), Hierarchical scheduling for diverse datacenter workloads, *in* 'Proceedings of the 4th Annual Symposium on Cloud Computing', SOCC '13, pp. 4:1–4:15.

Bhise, V. K. and Mali, A. S. (2013), Cloud resource provisioning for amazon ec2, *in* '2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)', pp. 1–7.

Bilal, K., Khan, S. and Zomaya, A. (2014), 'Trends and challenges in cloud data centers', *IEEE Cloud Computing* pp. 10–20.

Boloor, K., Chirkova, R., Salo, T. and Viniotis, Y. (2010), Heuristic-based request scheduling subject to a percentile response time sla in a distributed cloud, *in* '2010 IEEE Global Telecommunications Conference GLOBECOM 2010', pp. 1–6.

Bonald, T., Massoulié, L., Proutière, A. and Virtamo, J. (2006), 'A queueing analysis of max-min fairness, proportional fairness and balanced fairness', *Queueing Systems* pp. 65–84.

Boutaba, R., Cheng, L. and Zhang, Q. (2012), 'On cloud computational models and the heterogeneity challenge', *Journal of Internet Services and Applications* **3**.

Brams, S. J. and Weber, A. D. (1996), 'From cake-cutting to dispute resolution', *The American Mathematical* pp. 877–881.

Buyya, R., Ranjan, R. and Calheiros, R. N. (2009), Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities, *in* '2009 International Conference on High Performance Computing Simulation', pp. 1–11.

Calheiros, R., Ranjan, R., De Rose, C. and Buyya, R. (2009), 'Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services'.

Campos da Silva Filho, M., Oliveira, R., Monteiro, C., Inácio, P. and Freire, M. (2017), Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness, pp. 400–406.

Casas, E. and Tröltzsch, F. (2002), 'Second-order necessary and sufficient optimality conditions for optimization problems and applications to control theory', *SIAM Journal on Optimization* **13**, 406–431.

Cheung (n.d.).
URL: *http://www.mathcs.emory.edu/ cheung/Courses/558/Syllabus/11-Fairness/Fair.html*

CloudSim (2009), 'Cloudsim'.

Coluccia, A., D'Alconzo, A. and Ricciato, F. (2012), 'On the optimality of max-min fairness in resource allocation', *Annales des Télécommunications* **67**, 15–26.

Dandachi, G. (2015), 'Multihoming in heterogeneous wireless networks'.

Dolev, D., Feitelson, D., Halpern, J., Kupferman, R. and Linial, N. (2012), No justified complaints: On fair sharing of multiple resources, *in* 'Proceedings of the 3rd Innovations in Theoretical Computer Science Conference', pp. 68–75.

Farias, G., Lopes, R., Brasileiro, F. and Carvalho, M. (2020), Fair scheduling in cloud infrastructures withmultiple service classes.

Farley, B., Juels, A., Varadarajan, V., Ristenpart, T., Bowers, K. and Swift, M. (2012), More for your money: Exploiting performance heterogeneity in public clouds.

Friedman, E., Ghodsi, A. and Psomas, C. (2014), Strategyproof allocation of discrete jobs on multiple machines, *in* 'Proceedings of the Fifteenth ACM Conference on Economics and Computation', Association for Computing Machinery, p. 529–546.

Gautam, J. V., Prajapati, H. B., Dabhi, V. K. and Chaudhary, S. (2015), A survey on job scheduling algorithms in big data processing, *in* '2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)', pp. 1–11.

Gawali, M. and Shinde, S. (2018), 'Task scheduling and resource allocation in cloud computing using a heuristic approach', *Journal of Cloud Computing* **7**.

Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S. and Stoica, I. (2011), Dominant resource fairness: Fair allocation of multiple resource types, *in* 'Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation', pp. 323–336.

Ghodsi, A., Zaharia, M., Shenker, S. and Stoica, I. (2013), Choosy: Max-min fair sharing for datacenter jobs with constraints, pp. 365–378.

H, W. and P, V. (2014), Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation, *in* 'Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)', pp. 229–242.

Hamzeh, H., Meacham, S. and Khan, K. (2019), A new approach to calculate resource limits with fairness in kubernetes, *in* '2019 First International Conference on Digital Data Processing (DDP)', pp. 51–58.

Hamzeh, H., Meacham, S., Khan, K., Phalp, K. and Stefanidis, A. (2019), Ffmra: A fully fair multi-resource allocation algorithm in cloud environments, *in* '2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)', pp. 279–286.

Hamzeh, H., Meacham, S., Khan, K., Phalp, K. and Stefanidis, A. (2020), Mrfs: A multi-resource fair scheduling algorithm in heterogeneous cloud computing, *in* '2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)', pp. 1653–1660.

Hamzeh, H., Meacham, S., Virginas, B., Khan, K. and Phalp, K. (2019), Mlf-drs: A multi-level fair resource allocation algorithm in heterogeneous cloud computing systems, *in* '2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)', pp. 316–321.

Hansson, S. O. (2004), *Welfare, Justice, and Pareto Efficiency*, number 4, Springer, pp. 361–380.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S. and Stoica, I. (2011), Mesos: A platform for fine-grained resource sharing in the data center, *in* 'NSDI'.

Hosny, A., Faheem, H., Mahdy, Y. and Hedar, A.-R. (2016), 'Resource allocation algorithm for gpus in a private cloud', *International Journal of Cloud Computing* **5**, 45.

Jain, R., Chiu, D. M. and WR, H. (1998), 'A quantitative measure of fairness and discrimination for resource allocation in shared computer systems', *CoRR* .

Jain, S., Simon, H. U. and Tomita, E. (2005), *Algorithmic Learning Theory*, Springer.

Jakóbczyk, M. (2020), *Practical Oracle Cloud Infrastructure: Infrastructure as a Service, Autonomous Database, Managed Kubernetes, and Serverless.*

Jalal, K., Ioannis, L., George, K., Bhuvan, U. and Yiqiang, Z. (2017), 'An efficient and fair multi-resource allocation mechanism for heterogeneous servers'.

Jin, J., Wang, W.-H. and Palaniswami, M. (2009), 'Utility max–min fair resource allocation for communication networks with multipath routing', *Computer Communications* **32**(17), 1802–1809.

Jin, Y. and Hayashi, M. (2018), 'Trade-off between fairness and efficiency in dominant alpha-fairness family', pp. 391–396.

Joe-Wong, C., Sen, S., Lan, T. and Chiang, M. (2012*a*), Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework, *in* '2012 Proceedings IEEE INFOCOM', pp. 1206–1214.

Joe-Wong, C., Sen, S., Lan, T. and Chiang, M. (2012*b*), Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework, *in* '2012 Proceedings IEEE INFOCOM', pp. 1206–1214.

Khamse-Ashari, J. (2018), 'Fair and efficient resource management inheterogeneous cloud environments'.

Khamse-Ashari, J., Lambadaris, I., Kesidis, G., Urgaonkar, B. and Zhao, Y. (2017), Per-server dominant-share fairness (ps-dsf): A multi-resource fair allocation mechanism

for heterogeneous servers, *in* '2017 IEEE International Conference on Communications (ICC)', pp. 1–7.

Khamse-Ashari, J., Lambadaris, I., Kesidis, G., Urgaonkar, B. and Zhao, Y. (2018), A cost-efficient and fair multi-resource allocation mechanism for self-organizing servers, *in* '2018 IEEE Global Communications Conference (GLOBECOM)', pp. 1–7.

Khanna, A. and Sarishma (2015), Ras: A novel approach for dynamic resource allocation, *in* '2015 1st International Conference on Next Generation Computing Technologies (NGCT)', pp. 25–29.

Kolm, S. and See, H. (2002), *Justice and Equity*, Mit Press, MIT Press.
URL: *https://books.google.co.uk/books?id=HyctVz6tRbQC*

kube batch (n.d.).
URL: *https://github.com/kubernetes-sigs/kube-batch*

Kubilinskas, E. (2008), 'Design of multi-layer telecommunication networks'.

Li, J. and Xue, J. (2013), 'Egalitarian division under leontief preferences', *Economic Theory* pp. 597–622.

Li, L., Pal, M. and Yang, Y. R. (2008), Proportional fairness in multi-rate wireless lans, *in* 'IEEE INFOCOM 2008 - The 27th Conference on Computer Communications', pp. 1004–1012.

Li, W., Liu, X., Zhang, X. and Zhang, X. (2016), 'Multi-resource fair allocation with bounded number of tasks in cloud computing systems'.

Lin, F. and Su, J. (2017), 'Multi-layer resources fair allocation in big data with heterogeneous demands', *Wireless Personal Communications* **98**.

Lin, J.-C. and Lee, M.-C. (2016), 'Performance evaluation of job schedulers on hadoop yarn', *Concurrency and Computation: Practice and Experience* **28**(9), 2711–2728.

Liu, B., Y, Lin and Chen, Y. (2016), Quantitative workload analysis and prediction using google cluster traces, *in* '2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)', pp. 935–940.

Liu, H. and He, B. (2014), Reciprocal resource fairness: Towards cooperative multiple-resource fair sharing in iaas clouds, *in* 'SC '14: Proceedings of the International

Conference for High Performance Computing, Networking, Storage and Analysis', pp. 970–981.

Liu, X., Zhang, X., Cui, Q. and Li, W. (2017), Implementation of ant colony optimization combined with tabu search for multi-resource fair allocation in heterogeneous cloud computing, *in* '2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids)', pp. 196–201.

Liu, X., Zhang, X. and Li, W. (2017), 'Swarm optimization algorithms applied to multi-resource fair allocation in heterogeneous cloud computing systems', *Computing* **99**, 1–25.

Lopez-Pires, F. and Baran, B. (2017), 'Cloud computing resource allocation taxonomies', *International Journal of Cloud Computing* **6**, 238.

Lukka, K. (2003), *The Constructive Research Approach*, pp. 83–101.

Madej, A., Wang, N., Athanasopoulos, N., Ranjan, R. and Varghese, B. (2020), 'Priority-based fair scheduling in edge computing'.

Merkel, A. and Lohse, J. (2018), 'Is fairness intuitive? an experiment accounting for subjective utility differences under time pressure', *Experimental Economics* **0647**.

minikube (n.d.).
URL: *https://github.com/kubernetes-sigs/kube-batch*

Nace, D. and Pioro, M. (2006), 'A tutorial on max-min fairness and its applications to routing, load-balancing and network design'.

Namasudra, S., Roy, P. and Balusamy, B. (2017*a*), Cloud computing: Fundamentals and research issues, *in* '2017 Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM)', pp. 7–12.

Namasudra, S., Roy, P. and Balusamy, B. (2017*b*), Cloud computing: Fundamentals and research issues, *in* '2017 Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM)', pp. 7–12.

Negishi, T. (2006), 'Welfare economics and existence of an equilibrium for a competitive economy', *Metroeconomica* **12**, 92 – 97.

Nehru, E. I., Shyni, J. I. S. and Balakrishnan, R. (2016), Auction based dynamic resource allocation in cloud, *in* '2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)', pp. 1–4.

Nguyen, H., Shen, Z., Gu, X., Subbiah, S. and Wilkes, J. (2013), AGILE: Elastic distributed resource scaling for infrastructure-as-a-service, *in* 'Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)', USENIX, pp. 69–82.

Pahl, C. (2015), 'Containerization and the paas cloud', *IEEE Cloud Computing* **2**(3), 24–31.

Parkes, D., Procaccia, A. and Shah, N. (2015), 'Beyond dominant resource fairness: Extensions, limitations, and indivisibilities', *ACM Trans. Econ. Comput.* **3**, 3:1–3:22.

Poullie, P., Bocek, T. and Stiller, B. (2018), 'A survey of the state-of-the-art in fair multi-resource allocations for data centers', *IEEE Transactions on Network and Service Management* pp. 169–183.

Poullie, P. and Stiller, B. (2016), Cloud flat rates enabled via fair multi-resource consumption, *in* R. Badonnel, R. Koch, A. Pras, M. Drašar and B. Stiller, eds, 'Management and Security in the Age of Hyperconnectivity', Springer International Publishing, Cham, pp. 30–44.

Pscaer, J. (2018), 'Kubernetes in the enterprise'.
URL: *https://platform9.com/resource/the-gorilla-guide-to-kubernetes-in-the-enterprise/*

Psomas, C.-A. (2014), 'Strategyproof allocation of multidimensional tasks on clusters'.

Rajaram, B. (2014), 'Efficient, scalable, and fair read-modify-writes'.

Reiss, C., Tumanov, A., Ganger, G., Katz, R. and Kozuch, M. (2012), Heterogeneity and dynamicity of clouds at scale: Google trace analysis, *in* 'Proceedings of the Third ACM Symposium on Cloud Computing', SoCC '12, Association for Computing Machinery.

Saha, P., Beltre, A. and Govindaraju, M. (2019), 'Exploring the fairness and resource distribution in an apache mesos environment', *CoRR* **abs/1905.08388**.

Schimanski, S. and Hausenblas, M. (2018), 'Kubernetes deep dive: Api server'.
URL: *https://www.openshift.com/blog/kubernetes-deep-dive-api-server-part-1*

Sediq, A. B., Gohary, R. H. and Yanikomeroglu, H. (2012), Optimal tradeoff between efficiency and jain's fairness index in resource allocation, *in* '2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)', pp. 577–583.

Shah, N. (2017), Optimal social decision making, International Foundation for Autonomous Agents and Multiagent Systems.

Sharma, B., Chudnovsky, V., Hellerstein, J., Rifaat, R. and Das, C. (2011), Modeling and synthesizing task placement constraints in google compute clusters, *in* 'Proceedings of the 2Nd ACM Symposium on Cloud Computing', SOCC '11, ACM, pp. 3:1–3:14.

Singh, J. (2014), 'Study of response time in cloud computing', *International Journal of Information Engineering and Electronic Business* **6**, 36–43.

Singh, S. and Singh, N. (2016), Containers docker: Emerging roles future of cloud technology, *in* '2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)', pp. 804–807.

Srikant, R. (2004), *The Mathematics of Internet Congestion Control*.

Taddei, P. A. (2015), 'Design and development of acloudsim module to model andevaluate multi-resourcedependencie'.

Tahir, Y., Yang, S., Koliousis, A. and McCann, J. (2015), Udrf: Multi-resource fairness for complex jobs with placement constraints.

Tang, A., Wang, J. and Low, S. H. (2004), Is fair allocation always inefficient, *in* 'IEEE INFOCOM 2004', p. 45.

Tang, S., Niu, Z., He, B., Lee, B. and Yu, C. (2018), 'Long-term multi-resource fairness for pay-as-you use computing systems', *IEEE Transactions on Parallel and Distributed Systems* **29**, 1147–1160.

Vahora, S. and Patel, R. (2015), 'Cloudsim-a survey on vm management techniques', *IJARCCE* pp. 128–133.

Vaishnnave, M. P., Devi, K. and Srinivasan, P. (2019), A survey on cloud computing and hybrid cloud.

Vakilinia, S. (2015), 'Performance modeling and optimization of resource allocation in cloud computing systems'.

Vavilapalli, V, K., Murthy, A, C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., C, C., O'Malley, O., Radia, S., R, B. and Baldeschwieler, E. (2013), Apache hadoop yarn: Yet another resource negotiator, *in* 'Proceedings of the 4th Annual Symposium on Cloud Computing', ACM, pp. 5:1–5:16.

Wah, B. and wu, Z. (2002), The theory of discrete lagrange multipliers for nonlinear discrete optimization.

Wang, W., Li, B. and Liang, B. (2013), 'Dominant resource fairness in cloud computing systems with heterogeneous servers', *CoRR* **abs/1308.0083**.

Wang, W., Li, B., Liang, B. and Li, J. (2016), Multi-resource fair sharing for datacenter jobs with placement constraints, *in* 'SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis', pp. 1003–1014.

Xiao, Y., Huang, J., Yuen, C. and DaSilva, L. A. (2013), Fairness and efficiency tradeoffs for user cooperation in distributed wireless networks, *in* '2013 Proceedings IEEE INFOCOM', pp. 285–289.

Xu, X. and Yu, H. (2014), 'A game theory approach to fair and efficient resource allocation in cloud computing', *Mathematical Problems in Engineering* **2014**, 1–14.

Xue, N., Haugerud, H. and Yazidi, A. (2017), Towards a hybrid cloud platform using apache mesos, pp. 143–148.

Zahedi, S. M. and Lee, B. C. (2014*a*), Ref: resource elasticity fairness with sharing incentives for multiprocessors, *in* 'ASPLOS '14'.

Zahedi, S. M. and Lee, B. C. (2014*b*), Resource elasticity fairness with sharing incentives for multiprocessors, pp. 145–160.

Zhang, Q. and Boutaba, R. (2014), Dynamic workload management in heterogeneous cloud computing environments, *in* '2014 IEEE Network Operations and Management Symposium (NOMS)', pp. 1–7.

Zhang, Y., Wang, S. and ji, G. (2015), 'A comprehensive survey on particle swarm optimization algorithm and its applications', *Mathematical Problems in Engineering* **2015**, 1–38.

Zhao, L., Du, M. and Chen, L. (2018), 'A new multi-resource allocation mechanism: A tradeoff between fairness and efficiency in cloud computing', *China Communications* **15**, 57–77.

Zhao, L., Du, M., Lei, W., Chen, L. and Yang, L. (2018), Towards multi-task fair sharing for multi-resource allocation in cloud computing, Springer International Publishing, pp. 322–333.