

TIME AND COST-EFFICIENT RESOURCE  
ALLOCATION FOR REAL-TIME APPLICATION IN  
HIGH PERFORMANCE COMPUTING SYSTEMS

BY

MUHAMMAD SHUAIB QURESHI

A thesis submitted in fulfilment of the requirement for the  
degree of Doctor of Philosophy in Computer Science

Kulliyyah of Information and Communication Technology  
International Islamic University Malaysia

AUGUST 2021

## ABSTRACT


High Performance Computing (HPC) is the de-facto platform for deploying real-time applications due to the collaboration of large-scale resources operating in cross-administrative domains. HPC resource scheduling and allocation is a crucial issue in achieving efficient utilization of available resources, especially when resource-intensive applications have real-time deadlines and need data files replicated over the data storage resources. Such scheduling engages both computing and data storage resources to carry out application execution in a timely manner. Traditional approaches are sufficient only when data storage resources are coupled with the computing resources in HPC environment, since data is available at the computing resources for application execution. However, the said domain leaves gaps for deadline miss when data is transferred from remotely located data storage resources to the computing resources where application is being executed. The deadline miss mainly occurs due to the unavailability of the required data files, inadequate scheduling and allocation mechanism of the HPC resources. The problem becomes more complicated when some of the data files are pre-fetched while some post-fetched during application execution which usually results in delayed processing and in turn deadlines miss. The allocation of such resources by considering different optimization criteria such as makespan minimization, cost and energy efficiency, respecting application deadlines, etc. in the aforementioned scenario can be gracefully addressed by designing a scheduling strategy which can result in improved resources utilization while predicting application feasibility. It has always been of interest to the research community to pose the abovementioned situation to determine if the existing scheduling theory and resource allocation strategies are mature enough to accommodate the challenges presented with the emergence of the latest HPC platforms. In this thesis, we explore and analyze the existing resource-allocation techniques for scheduling real-time applications with temporal constraints on HPC platforms (grid, cloud, edge, fog, and multicore systems). This study further compares the resource allocation mechanisms based on different performance parameters and based on existing gaps, a model is proposed which predicts the application schedulability by analyzing time and data constraints before actually dispatching the application to the HPC resources. The main advantage of the prediction-based model is to save time by declining further analysis on unsuitable resources which improve resource utilization by considering application workload in advance. Furthermore, this research thesis devises time and cost-efficient variants of HPC resource allocation with provably correct formulations to cope with the aforementioned problems so that both the user and real-time application constraints are respected. The most celebrated results affirm the supremacy of the proposed techniques in obtaining the desired level of service.

## خلاصة البحث

الحوسبة عالية الأداء هي عبارة عن منصة تقوم بالتخصيص المنسق للموارد واسعة النطاق على التطبيقات الحالية التي يتم تشغيلها ضمن المجالات ذات الإدارات المتداخلة. تعتبر عملية جدولة وتخصيص الموارد ضمن مجال الحوسبة عالية الأداء مسألة في غاية الأهمية حيث ترتبط بتحقيق الاستغلال الأمثل للموارد المتاحة، وخصوصاً عندما تكون هناك مواعيد محددة للتطبيقات ذات الاستهلاك الكثيف للموارد، والتي تتطلب استنساخ الملفات عبر موارد تخزين البيانات. تقوم الجدولة بالتنسيق بين عمليات الحوسبة وبين موارد تخزين البيانات بما يضمن تنفيذ التطبيق في الوقت المحدد. تعتبر الأساليب التقليدية كافية فقط في حالة كان هناك اقتران بين موارد تخزين البيانات وبين موارد عمليات الحوسبة ضمن بيئة الحوسبة عالية الأداء، بحيث يتم توفير البيانات اللازمة لتنفيذ التطبيق ضمن موارد الحوسبة. إلا أن الأساليب المذكورة تؤدي إلى حدوث فجوات تسبب في تفويت المواعيد المحددة وخصوصاً عندما يتم نقل البيانات من موارد التخزين النائية إلى موارد الحوسبة التي يتم فيها تنفيذ التطبيقات. بشكل عام، فإنه يتم تفويت المواعيد المحددة بسبب عدم توفر ملفات البيانات المطلوبة، وبسبب عدم كفاءة عمليات الجدولة وآليات تخصيص الموارد ضمن بيئة الحوسبة عالية الأداء. كما تصبح المشكلة أكثر تعقيداً عندما يتم جلب ملفات البيانات مسبقاً وجلب البعض الآخر لاحقاً أثناء عملية تنفيذ التطبيق، حيث يؤدي ذلك في العادة إلى تأخر عمليات المعالجة، والذي يؤدي بدوره إلى تفويت المواعيد المحددة. يمكن معالجة مشكلة تخصيص الموارد من خلال إعادة النظر في المعايير المختلفة للتحسين بما في ذلك: تقليل زمن التنفيذ الكلي، وكفاءة التكلفة والطاقة، والالتزام بالمواعيد المحددة، وغيرها، ضمن السيناريو السابق، حيث يمكن معالجة ذلك بشكل فعال من خلال إعادة تصميم استراتيجية الجدولة بحيث تعمل على الاستغلال الأمثل للموارد، وفي ذات الوقت، تقوم بالنتوء بجداول التطبيق. لطالما كان هناك اهتمام من الباحثين بالجانب أعلاه وذلك لتحديد ما إذا كانت نظرية الجدولة الحالية واستراتيجيات تخصيص الموارد فعالة بما فيه الكفاية لمواجهة التحديات التي برزت مع ظهور منصات الحوسبة عالية الأداء. سنقوم في هذه الأطروحة باستكشاف وتحليل الأساليب الحالية لتخصيص الموارد وجدولة التطبيقات الحالية ذات القيود الزمنية على منصات الحوسبة عالية الأداء بما في ذلك (الأنظمة الشبكية، والسحابية، والضبابية، والطرفية، والأنظمة متعددة الأنوية). كما تقوم هذه الدراسة أيضاً بالمقارنة بين آليات تخصيص الموارد القائمة على أساس معايير الأداء المختلفة، والقائمة على أساس الفجوات، حيث يعمل هذا النموذج المقترح على التنبؤ بالجدول من جدولة التطبيق عن طريق تحليل القيود الزمنية والبياناتية قبل إرسال التطبيق فعلياً إلى موارد الحوسبة عالية الأداء. تتمثل الميزة الرئيسية للنموذج القائم على التنبؤ في توفير الوقت من خلال رفض طلبات التحليل الإضافية للموارد غير المناسبة، حيث يعمل ذلك على تحسين استغلال الموارد من خلال النظر مقدماً إلى حجم العبء المجدول على التطبيق. علاوة على ذلك، تعمل هذه الأطروحة على ابتكار متغيرات لتخصيص موارد الحوسبة عالية الأداء ذات كفاءة أكبر من ناحية الوقت ومن ناحية التكلفة، وذات إعدادات مناسبة للتعامل مع المشاكل الآتية الذكر، وبحيث يتم الأخذ في الحسبان كلاً من القيود الآتية للمستخدم والقيود الآتية للتطبيق. تشير أبرز النتائج المنشورة إلى تفوق الأساليب المقترحة في تحقيق المستوى المطلوب من الخدمة.

## APPROVAL PAGE

The thesis of Muhammad Shuaib Qureshi has been approved by the following:



---

Asadullah Shah  
Supervisor

---

Amelia Ritahani Bt. Ismail  
Co-Supervisor

---

Rizal Bin Mohd. Nor  
Co-Supervisor

---

Amir 'Aatieff Bin Amir Husin  
Internal Examiner

---

Ali Selamat  
External Examiner

---

Radwan Jamal  
Chairman

## DECLARATION

I hereby declare that this thesis is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted as a whole for any other degrees at IIUM or other institutions.

Muhammad Shuaib Qureshi



Signature .....

Date .....

**INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA**

**DECLARATION OF COPYRIGHT AND AFFIRMATION OF  
FAIR USE OF UNPUBLISHED RESEARCH**

**TIME AND COST-EFFICIENT RESOURCE ALLOCATION  
FOR REAL-TIME APPLICATION IN HIGH PERFORMANCE  
COMPUTING SYSTEMS**

I declare that the copyright holders of this thesis are jointly owned by the student and IIUM.

Copyright © 2020 Muhammad Shuaib Qureshi and International Islamic University Malaysia. All rights reserved.

No part of this unpublished research may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior written permission of the copyright holder except as provided below

1. Any material contained in or derived from this unpublished research may be used by others in their writing with due acknowledgement.
2. IIUM or its library will have the right to make and transmit copies (print or electronic) for institutional and academic purposes.
3. The IIUM library will have the right to make, store in a retrieved system and supply copies of this unpublished research if requested by other universities and research libraries.

By signing this form, I acknowledged that I have read and understand the IIUM Intellectual Property Right and Commercialization policy.

Affirmed by Muhammad Shuaib Qureshi



.....  
Signature

.....  
Date

## ACKNOWLEDGEMENTS

I offer heartiest “Darood-O-Salam” to Holy Prophet “MUHAMMAD” (Peace Be Upon Him). I am grateful to almighty ALLAH who is merciful and beneficent, and who enabled me to work on this research successfully. Accomplishment of a research thesis requires the help of many people who steer, guide, give confidence and help you. First, I would like to express my sincere gratitude to my supervisor, “*Prof. Dr. Asadullah Shah*” for his esteemed supervision, encouragement and guidance for successful completion of this research work. I will be forever grateful to him for being a great mentor in my professional life as well as in my personal life. He made my time a precious and joyful memory. I am thankful to my co-supervisors, *Dr. Amelia Ritahani Bt. Ismail and Dr. Rizal Bin Mohd. Nor* for their useful comments, suggestions and guidance. I am thankful to my colleagues, friends and family members who encouraged me to complete this work. My special thanks to my brother *Dr. Muhammad Bilal Qureshi*, who encouraged me at every step of my research work. I am really grateful for his sincere and unconditional help throughout my educational career. Last but not the least, special thanks to my parents for their unconditional, unwavering, and untiring support, wishes, and encouragement. They have been my strongest motivation throughout my thesis.

Muhammad Shuaib Qureshi

# TABLE OF CONTENTS

Abstract.....	ii
Abstrac Arabic .....	iii
Approval Page.....	iv
Declaration.....	v
Copyright Page.....	vi
Acknowledgements.....	vii
Table of Contents.....	viii
List of Tables .....	xi
List of Figures .....	xii
List of Abbriviations .....	xiv
Notations And Description .....	xv
<b>CHAPTER ONE: INTRODUCTION.....</b>	<b>1</b>
1.1 Background of the Study .....	1
1.2 Problem Statement.....	5
1.3 Research Objectives .....	6
1.4 Ultimate Research Questions.....	7
1.5 Significance/Expected Outcomes of the Research .....	8
1.6 Research Design/Methodology.....	10
1.6.1 Literature Survey.....	11
1.6.2 Problem Formulation .....	11
1.6.3 Answers to the Research Questions .....	11
1.6.4 Analysis and Validation .....	11
<b>CHAPTER TWO:LITERATURE REVIEW.....</b>	<b>12</b>
2.1 Introduction .....	12
2.2 Real-time Application Model .....	13
2.3 Brief Comparison with the Existing Surveys .....	15
2.4 Chapter Organization.....	17
2.5 HPC Resource Allocation Problem for RT Services.....	17
2.6 Evaluation Criteria.....	19
2.7 Resource Allocation Schemes for Real-Time Services in Grid Computing .....	30
2.7.1 Dynamic Voltage Scaling .....	31
2.7.2 Real-Time Data-Intensive Tasks Allocation Technique .....	33
2.7.3 Energy Efficient Genetic-based Scheduling .....	33
2.8 Resource Allocation Schemes for Real-Time Services in Cloud Computing .....	34
2.8.1 Partition Problem-based Dynamic Provisioning and Scheduling Scheme .....	36
2.8.2 Dynamic Scheduling Bag of Tasks .....	38
2.8.3 Heuristic-based Resource Allocation Schemes.....	38
2.8.4 Data-aware Resource Allocation Scheme.....	40
2.8.5 Earliest Finish Time Duplication Approach .....	41



2.8.6 Task Scheduling and Load Balancing Technique.....	43
2.8.7 Proactive and Reactive Scheme .....	44
2.8.8 Allocation-aware Task Scheduling .....	45
2.8.9 Bandwidth-aware Resource Allocation .....	46
2.8.10 Bat Approach for Resource Allocation.....	47
2.8.11 Developmental Genetic Programming.....	48
2.8.12 Federation-based Resource Allocation Scheme.....	49
2.8.13 Adaptive Genetic Approach.....	51
2.8.14 Dynamic Fault-tolerant Elastic Scheduling (DFES).....	52
2.8.15 Earliest Deadline First Greedy Approach .....	53
2.8.16 Deadline-oriented VM Allocation Approach.....	54
2.8.17 Approximate Computation-based Resource Allocation Scheme .....	56
2.8.18 Periodic Server Scheduling Scheme .....	57
2.8.19 Heuristic-based Earliest Deadline First Scheme.....	58
2.8.20 Application-directed Check Pointing and Approximate Computation Scheme.....	59
2.8.21 Earliest Deadline First and Unfair Semi-Greedy Approach .....	59
2.8.22 Dynamic Proactive Reactive Scheduling.....	60
2.8.23 Energy-aware Resource Allocation .....	60
2.8.24 Bag of Tasks Scheduling with Approximate Computation .....	61
2.8.25 Green Cloud Scheduling Approach .....	62
2.8.26 Fuzzy Dominance Sort-based Resource Allocation Technique .....	63
2.8.27 Best Fit with Imprecise Computations.....	63
2.8.28 Hybrid Genetic and Cuckoo Search (HGCS) Algorithm.....	65
2.9 Resource Allocation Schemes for Real-Time Services in Edge Computing .....	65
2.9.1 Resource Matching-based Allocation Scheme .....	68
2.10 Resource Allocation Schemes for Real-Time Services in Fog Computing .....	68
2.10.1 Hybrid Earliest Deadline First Approach.....	70
2.10.2 Tasks Buffering and Offloading Policy .....	71
2.11 Resource Allocation Schemes for Real-Time Services in Multicore Systems.....	73
2.11.1 Rate Monotonic Scheduling with Reduced Priority Levels Approach .....	73
2.11.2 Hybrid Cuckoo Search-based Algorithm.....	73
2.11.3 Online Accrued Scheduling Scheme.....	74
2.11.4 Least Feasible Speed (LFS) Technique .....	74
2.11.5 Load Balancing by Tasks Splitting and Tasks Shifting Strategy .....	75
2.11.6 Compatibility-aware Task Partitioning Scheme .....	76
2.11.7 Simple Combined Resource Usage Partitioning.....	77
2.11.8 Enhancing Shared Cache Performance-based Approach.....	78
2.11.9 Large Time Demand Analysis Technique .....	79
2.12 Chapter Summary .....	79

<b>CHAPTER THREE: PREDICTION-BASED RESOURCE ALLOCATION MODEL FOR REAL-TIME TASKS .....</b>	<b>81</b>
3.1 Introduction .....	81
3.2 Proposed Resource Allocation Model .....	82
3.3 Mathematical Modeling and Problem Formulation.....	83
3.3.1 Proposed Task and Resource Model .....	83
3.3.2 Basic Task Model.....	84
3.3.3 Modified Task Model.....	85
3.3.4 Metatask $\Gamma$ .....	85
3.3.5 Offline Prediction Analyzer .....	85
3.3.5.1 Positive Negative Points Set for Task $i$ .....	86
3.3.5.2 Resource Demand Calculator .....	86
3.3.5.3 Execution Time of Task $i$ on Computing Resource $r$ .....	86
3.3.5.4 Resource Ranking Function .....	87
3.3.5.5 Tasks Grouping .....	88
3.3.5.6 Priority Assigner.....	88
3.3.5.7 Objective Function .....	88
3.3.5.8 Resource Model.....	89
3.4 Chapter Summary .....	89
 <b>CHAPTER FOUR: EFFICIENT RESOURCE ALLOCATION TECHNIQUE FOR REAL-TIME DATA-INTENSIVE TASKS IN CLOUD COMPUTING SYSTEMS .....</b>	 <b>90</b>
4.1 Introduction .....	90
4.2 Task, Resource and Cost Models .....	93
4.2.1 Task and Resource Model.....	93
4.2.2 Data Files Model.....	95
4.2.3 Tasks Grouping .....	96
4.2.4 Cost Model.....	102
4.3 Time and Cost-Efficient Scheduling Algorithm.....	103
4.4 Performance Evaluation .....	107
4.4.1 Experimental Setup .....	107
4.4.2 Performance Metrics .....	110
4.5 Results and Discussion .....	111
4.5.1 Effect of Data Files Transfer on Performance .....	117
4.5.2 Impact on Resource Utilization.....	119
4.6 Chapter Summary .....	120
 <b>CHAPTER FIVE: CONCLUSION AND FUTURE WORK.....</b>	 <b>122</b>
5.1 Conclusion .....	122
5.2 Theoretical, Practical and Methodological Contributions .....	123
5.3 Future Work.....	125
 <b>REFERENCES.....</b>	 <b>127</b>

## LIST OF TABLES

Table 2.1 Comparison of the state-of-the-art HPC resource allocation schemes for real-time applications	25
Table 4.1 Simulation parameters settings	110
Table 4.2 Tasks groups	114

## LIST OF FIGURES

Figure 1.1 High-level view of resource allocation in HPC system	4
Figure 1.2 Proposed resource allocation scenario.	7
Figure 2.1 Real-time application taxonomy	13
Figure 2.3 Distributed HPC RA schemes taxonomy for real-time systems	20
Figure 2.4 Grid computing environment	30
Figure 2.5 Taxonomy of grid systems	31
Figure 2.6 High level scenario of dynamic voltage scaling scheme in grid computing	32
Figure 2.7 RDTA approach	33
Figure 2.8 General architecture of cloud computing environment	35
Figure 2.9 Cloud computing taxonomy	36
Figure 2.10 Workflow of PPDPS scheme	37
Figure 2.11 Flowchart of DSB technique	39
Figure 2.12 Task allocation process of heuristic-based allocation scheme	40
Figure 2.13 Overview of the data-aware resource allocation scheme	41
Figure 2.14 Workflow of EFTD RA scheme	42
Figure 2.15 Task scheduling and load balancing technique	44
Figure 2.16 Working of PRS RA scheme	45
Figure 2.17 High level view of ATS algorithm	46
Figure 2.18 Bandwidth-aware RA scheme workflow	47
Figure 2.19 Working of BAT algorithm	48
Figure 2.20 Flowchart of Developmental Genetic algorithm	49
Figure 2.21 Federation-based RA scheme	51
Figure 2.22 Simplified flowchart of AGA scheme	53

Figure 2.23 Workflow of DFES algorithm	54
Figure 2.24 EDF Greedy task allocation procedure	55
Figure 2.25 Deadline-oriented VM allocation approach	56
Figure 2.26 Flow of approximate computation-based RA scheme	57
Figure 2.27 Workflow of BoT with approximate computation scheduling approach	64
Figure 2.28 General architecture of edge computing	67
Figure 2.29 Taxonomy of edge computing	67
Figure 2.30 General architecture of fog computing	69
Figure 2.31 Taxonomy of fog computing	70
Figure 2.32 General process of hybrid EDF approach	71
Figure 2.33 Task buffering and offloading scenario	72
Figure 2.34 High level flow of online accrued scheduling scheme	76
Figure 2.35 Compatibility-aware tasks partitioning scheme	77
Figure 3.1 Proposed prediction-based resource allocation model	84
Figure 3.2 Resource rank calculation	87
Figure 4.1 Tasks grouping taxonomy	96
Figure 4.2 RM scheduling of $\gamma$ in Example 1	102
Figure 4.3 Average makespan	115
Figure 4.4 Average cost	116
Figure 4.5 Data transfer time	118
Figure 4.6 Effect on resource utilization	120

## LIST OF ABBRIVIATIONS

HPC	High Performance Computing
QoS	Quality of Service
RA	Resource Allocation
DVS	Dynamic Voltage Scaling
RDTA	Real-time Data-intensive Tasks Allocation
GA-SS	Steady State Genetic Algorithm
GA-ST	Struggled Genetic Algorithm
GA-EG	Elitist Generational Genetic Algorithm
PPDPS	Partition Problem based Dynamic Provisioning and Scheduling
DSB	Dynamic Scheduling of Bag of Tasks
MAHP	Modified Analytic Hierarchy Process
BATS	Bandwidth Aware Tasks Scheduling
LEPT	Longest Expected Processing Time
AGA	Adaptive Genetic Algorithm
EFTD	Earliest Finish Time Duplication
PRS	Proactive and Reactive Scheduling
ATS	Allocation-aware Task Scheduling
DGP	Developmental Genetic Programming
DCLS	Dynamic Cloud List Scheduling
DCMMS	Dynamic Cloud Min-Min Scheduling
EDF	Earliest Deadline First
DVFS	Dynamic Voltage Frequency Scaling
HGCS	Hybrid Genetic and Cuckoo Search
PTPS	Purely Time-driven Periodic Server
WCPS	Work-Conserving Periodic Server
CRPS	Capacity Reclaiming Periodic Server
USG	Unfair-Semi Greedy
EDZL	Earliest Deadline until Zero-Laxity
EARH	Energy Aware Rolling Horizon
GCSM	Green Cloud Scheduling Model
FDHEFT	Fuzzy Dominance sort based Heterogeneous Earliest Finish Time
DA-EDF	Data Aware – Earliest Deadline First
EDA-EDF	Enhanced Data Aware – Earliest Deadline First
RS	Resource Scheduling
RM	Resource Matching/Rate Monotonic
HCS	Hybrid Cuckoo Search
LFS	Least Feasible First
CATP	Compatibility Aware Task Partition
G-CATP	Group-wise Compatibility Aware Task Partition
SCRUP	Simple Combined Resource Usage Partitioning
ENCAP	Enhancing Shared Cache Performance
TDA	Time Demand Analysis
DEFT	Dynamic Fault-Tolerant Elastic

## NOTATIONS AND DESCRIPTION

$T$	Task set
$r_k$	Release time of task $k$
$e_k$	Execution time of task $k$
$d_k$	Deadline of task $k$
$CP_y$	Computing power of resource $y$
$PNP$	Positive-negative points set
$DF_k$	Data files set required by task $k$
$EET_{ky}$	Execution time of task $k$ on resource $y$
$Y_x$	Tasks group $x$
$GU_{Y_x}$	Group utilization of tasks group $x$
$TU_i$	Utilization of task $i$
$CD$	Compute and storage resource pair
$CR$	Computing resource set
$t_P$	Positive point
$t_N$	Negative point
$TT$	Total execution time
$DR$	Data storage resource set
$\mathbb{R}_w$	Response time of the storage resource $w$
$f_{kz}$	File $z$ needed by task $k$
$FT_{f_{kz}}$	Transfer time of the file $f_{kz}$
$card(T)$	Cardinality of task set $T$
$dr_w$	Data storage resource $w$

# **CHAPTER ONE**

## **INTRODUCTION**

### **1.1 BACKGROUND OF THE STUDY**

High Performance Computing (HPC) is an attractive platform for both academic research and ICT trade to execute computational-intensive applications which need powerful resources for generating celebrated results. Many of such applications are time critical which needs in time response for completion. Such applications are known as real-time applications. In real-time applications, the correctness of the system not merely depends on the produced results but the time in which these results are obtained. Real-time systems are characterized by some parameters like computation time, period, and deadline. The computation time is the system's execution demand for the computing resource. Each real-time system generates infinite instances called as jobs. A job is generated after a specific time interval called as period. The time before which the application should complete its execution is called as deadline. Depending on the consequences of the missed deadlines, the real-time systems are broadly categorized into hard and soft real-time systems (Qureshi, Alrashed, Min-Allah, Kolodziej & Arabas, 2015). In hard real-time systems, a deadline meeting is the most critical constraint. Examples of such systems include railway switching system, air traffic control system, nuclear plant control system, and military system. The hard-real-time systems must respect the deadline constraints. In soft real-time systems, there is a gap for deadlines miss which may not result in catastrophic behavior but system's performance degradation. Examples of soft real-



time systems include automated teller machines, virtual reality, multimedia systems, interactive computer games, mobile robotics, and telecommunication networks.

A real-time system is composed of concurrent programs known as tasks (Laplante, 2004). A real-time task can be defined as an executable entity of work that is characterized by deadline and execution time which is the maximum estimated time needed by a processor to complete the task. This time is termed as worst-case execution time. By considering the aforementioned two basic characteristics, a real-time task  $i$  is denoted by  $\tau_i(c_i, d_i)$ , where  $c_i$  and  $d_i$  represent execution time and deadline of the task  $i$  respectively. Since, real-time systems are time sensitive systems, so scheduling such systems got massive popularity in the literature and numerous algorithms for different scenarios have been proposed (Zhang, Tian, Fidge, & Kelly, 2016). Some of the scheduling algorithms guarantee in advance that the application constraints will be met during execution. Such algorithms are known as static scheduling algorithms. In static priority assignment algorithms, the tasks priorities once set remain constant throughout the execution of the task. The well-known static priority assignment algorithm is rate-monotonic (RM) algorithm. The RM algorithm prioritize tasks on the basis of their rates: the higher is the rate, the higher is the priority. The rate of the task is inversely proportional to the period of the task i.e.,  $rate = \frac{1}{period}$ . The other class of algorithms prioritizes application during execution. Such algorithms are known as dynamic scheduling algorithms. Both algorithms classes characterize tasks with additional parameters, which are used to analyze performance of the system.

The HPC paradigm is attractive platform for deploying real-time applications mainly for three reasons:

1. the time sensitive nature of the real-time application requires parallel processing on distributed powerful resources to generate results in a timely manner,
2. the concurrent execution on many high-speed interconnected nodes as compared to a single powerful CPU is economical to efficiently achieve the desired level of performance, and
3. the distributed systems are highly reliable in cases of system failure.

But the existing HPC platforms still face many challenges due to the heterogeneous nature of distributed resources like predicting system behavior in peak load conditions (when all tasks occur at critical instant), completing tasks with minimum total execution time and user budget, proper load balancing on resources, on-time resource provisioning, dealing with task and resource heterogeneity, fault tolerance, a-priori time management requirements, and so on. Such challenges pave the way for further investigations and need to be addressed properly by developing adequate scheduling mechanisms to execute real-time applications within deadlines while meeting user QoS criteria. A proper resource allocation (RA) mechanism improves performance of all HPC classifications (Hussain, Malik, & Khan, et al., 2013; Qureshi, Dehnavi, & Min-Allah, et al., 2014).

Currently, task scheduling and resource allocation techniques attracted researchers towards HPC platforms considering diverse optimization criteria of virtual machines (VMs) renting cost, makespan minimization, QoS maximization, energy efficiency, and so on according to predefined agreed SLAs (Liu et al., 2015; Sangwan et al., 2016; Awad. A et al., 2015; Chen. H et al., 2015; Wu. X et al., 2013; Panda et al., 2015; Satish & Reddy, 2018). In this thesis, we use system, application, task, and job interchangeably.

A generalized RA scenario in HPC systems is represented in Figure 1.1 High-level view of resource allocation in HPC system A user submits request for RA to the broker which finds the resources status form the HPC information service directory. This directory holds information about resources. Based on this information, a large pool of resources is searched and resources which fulfil user QoS criteria are selected for application execution. The user application is submitted to the selected resources for execution. After successful execution of the application, the results are returned to the user.

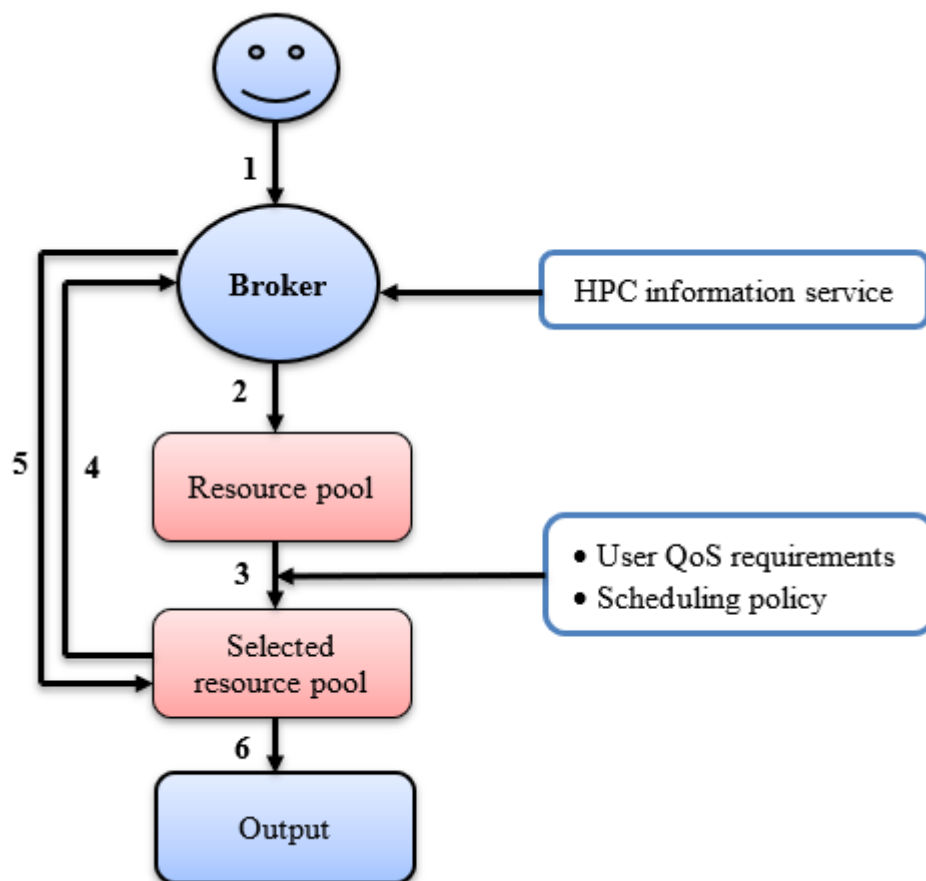


Figure 1.1 High-level view of resource allocation in HPC system

## 1.2 PROBLEM STATEMENT

It is observed from literature that most of the existing solutions to the real-time task's allocation problem in HPC domain provide room for deadline miss when such applications need data files for processing. This type of applications is called as data-intensive real-time applications. The data and communicational aspects between computing and storage resources are unaccounted in the existing resource optimization (scheduling and mapping) settings when data storage resources are located remotely from the computational resources. The feasibility testing of the real-time tasks under fixed priority scheduling technique has always been challenging in data-intensive real-time applications when some of the required data files are pre-fetched while some post-fetched during tasks execution. The literature also lacks comprehensive mechanism for predicting tasks feasibility prior to execution on different HPC resources when time and data constraints are considered. This gap provides opportunity for tasks scheduling on non-feasible resources and hence deadlines miss. From data files processing aspects, it has also been of interest to include the data files transfer time in deciding tasks feasibility on computational resources. The research community focused on the deadline's fulfilment concern of the real-time applications execution and ignores the user budget constraints.

The aim of this research is to develop a novel and fine-tuned resource scheduling and allocation policy for real-time application on HPC resources to cope with the aforementioned problems. We believe that this attempt will result in remarkable contributions towards plethora of existing real-time systems scheduling literature. Based on the nature of research work and above discussed problems, it is mainly divided into three modules (portrayed in Figure 1.2) with clear objectives.

In Module – I, we study and analyze the real-time resource allocation (RA) strategies in HPC domain. The performance of each RA strategy is evaluated on the basis of common parameters. The Module – I describes and pictorially represents each studied strategy in detail which helps in understanding working of each mechanism.

Module – II devises a prediction-based resource allocation model for real-time data-intensive application. The developed model checks the feasibility of tasks before actually allocating and dispatching tasks to the computing resources. The research work in Module – II devises a new scheduling model for scheduling real-time application to enhance schedulability of tasks by considering different optimization criteria.

In Module – III, we propose a resource allocation strategy for real-time data-intensive tasks by ranking computational resources and classifying tasks into groups that guarantee tasks execution with minimum possible time and cost while deadlines are kept intact. The ranking technique helps in selecting the most appropriate resources for application scheduling. The obtained results affirm the supremacy of the proposed technique over the existing counterparts.

### **1.3 RESEARCH OBJECTIVES**

The aims and objectives of this research are:

1. To collect and pictorially present the existing resource scheduling and allocation techniques in different HPC systems (grid, cloud, fog, edge, and multicore) at one place under the umbrella of real-time literature by considering common performance parameters.
2. To propose a two-stage model where the first stage predicts the feasibility of real-time application before actually dispatching to the HPC resources

and, the second stage schedule the application on the feasible resources by considering objective function.

3. To propose a time and cost-efficient resource allocation strategy for data-intensive real-time application in the HPC system which reduces the priority levels.

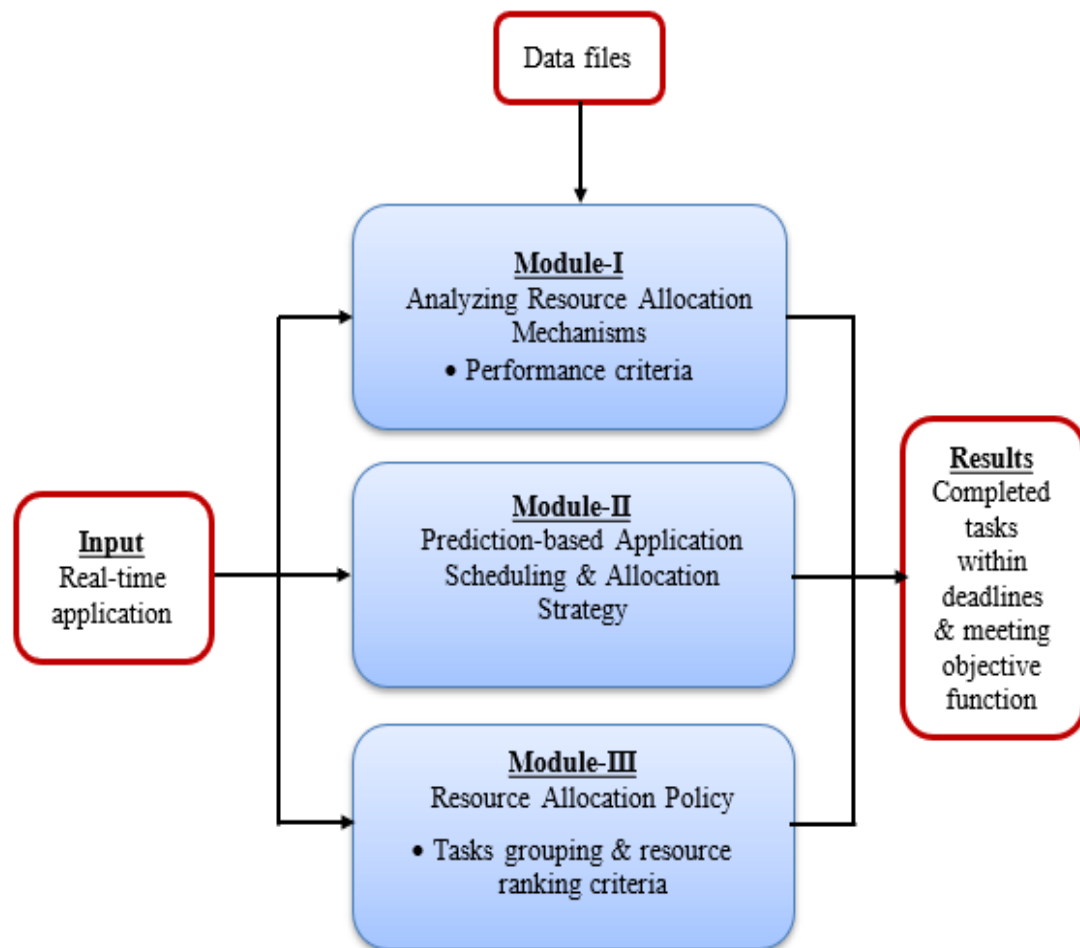


Figure 1.2 Proposed resource allocation scenario.

#### 1.4 ULTIMATE RESEARCH QUESTIONS

The HPC platforms offer celebrated capabilities if the encompass distributed resources are managed in an efficient way across heterogeneous environments. The existing

literature is still immature in modeling the HPC resources in a comprehensive manner which leave some questions open ended that need to be answered. The ultimate questions that the proposed resource allocation research addresses are:

1. How HPC paradigm can help more efficiently in scheduling and processing real-time applications within deadlines as compared to other platforms?
2. What issues can arise during real-time application scheduling and mapping, specially, when both computing and storage resources are heterogeneous in nature and involved in resource allocation strategy?
3. How can it be verified in advance whether the real-time application constraints will be met during execution or not?
4. How the resource allocation strategy allocates resources and executes real-time data-intensive tasks in minimum possible execution time and limited budget constraints while respecting tasks deadlines?
5. What effects the transfer time put on the tasks total execution time and cost when data files are replicated on remotely located storage resources?

### **1.5 SIGNIFICANCE/EXPECTED OUTCOMES OF THE RESEARCH**

The HPC is resources rich paradigm which is considered the most promising platform for deploying real-time applications. The increasing complexity of the real-time applications make HPC model a handy candidate to place such systems. The accuracy of generated results deliberately depends on the resource allocation mechanism. An efficient resource allocation mechanism can contribute to the best performance of the resources in handling time critical applications in an eminent fashion, especially, when resources are heterogeneous in nature.

This research delivers broadly to the scheduling theory that opens new dimensions for extending the existing mechanisms in broad ways which contributes to the society. Schedulers and resource allocation mechanisms are considered to be the core components of operating systems, parallel systems, and real-time embedded systems. It helps researchers how to take advantage of powerful resources in executing time-conscious applications in an efficient way such that the deadlines are respected. Moreover, the proposed model is flexible in order to adapt to the future paradigms and multiple domains by future researchers and scientists to immensely improve the throughput and reliability of the systems.

This research work advances the current state-of-the-art of real-time scheduling theory using HPC platforms as follows.

1. *Identification of the positive and negative scheduling points*

The proposed work identifies the impact of negative points on deadlines miss during scheduling real-time application. Such points can be disjointed from the positive points in a set which need not to be checked during feasibility testing of the real-time application on HPC resources. This mechanism can help in reducing application completion time which in turn ensures fulfilling the deadlines.

2. *Specification of criteria for the selection of computing and storage resources for data-intensive real-time application*

This research accumulates real-time application which needs data-files for complete execution. The data-files are replicated on distributed data storage resources connected to the computing resources by network links. The proposed technique designs a criterion for the selection of storage and



computing resources in a way such that the resources are engaged for a short duration of time while respecting application deadlines.

3. *Designing a model for efficient utilization of resources*

HPC resource is an important entity that needs efficient utilization for enjoying its maximum capacity. The proposed study helps in developing effective mechanism for utilizing full capabilities of computing resources in minimum price.

4. *Developing a mechanism for selecting least cost HPC resources for executing real-time data-intensive application*

The advent of big data handling is a challenging task, specifically when application has real-time deadlines. The anticipated model helps in developing a mechanism for selecting least cost computing and storage resources for executing application with real-time and data constraints. The model considers both local and remote storage resources and helps in reducing data transfer time which ultimately reduces application processing time.

## **1.6 RESEARCH DESIGN/METHODOLOGY**

This research work adopts quantitative approach which constructs mathematical models, theorems, simulations and quantitative evaluations. Quantitative research approach is a strong technique in validating proposed research claims. The research process is conducted by gradually following the undermentioned phases.

### **1.6.1 Literature Survey**

In this phase, the existing literature consisting of latest articles from well-known research journals, conferences, books, and pedagogical is studied in order to know the HPC resource allocation domain and its different dimensions. During this activity, the originality of the found research problems and the proposed solution method is ensured.

### **1.6.2 Problem Formulation**

The identified problems in literature survey phase are mathematically formulated in a proper format and the answers to the research questions are prepared by the proposed research work.

### **1.6.3 Answers to the Research Questions**

In this phase, the collected data is scrutinized to find answers and solutions to the research questions formulated in the previous phases.

### **1.6.4 Analysis and Validation**

In the analysis and validation phase, synthetic data sets are generated to evaluate and validate the proposed solution. The synthetic data sets are generated using model presented by Bini, & Buttazzo, (2005). This is the most authentic and followed model in the existing literature. The computing and storage resources are modeled, and simulations are carried out using Matlab (2019 version). Matlab is a strong tool for complex mathematical modeling. The obtained results are compared, and cross checked against the existing counterparts to validate and evaluate the performance of the proposed mechanism.

## **CHAPTER TWO**

### **LITERATURE REVIEW**

#### **2.1 INTRODUCTION**

The High-Performance Computing (HPC) paradigm paves the way to the large number of resources with high computing power and storage capabilities distributed across a network. The HPC platform is focused by the research community, mainly for three basic reasons: (a) the parallel nature of applications, (b) the need of data distribution and communication with other nodes through communication channels, and (c) the need for continuous availability and reliability of the resources. These facilities motivate the users to use distributed HPC systems for scalability, reliability, availability, exchanging and sharing of information, and achieving performance efficiency with low cost and high quality (Hussain et al., 2013; Qureshi et al., 2014; Y. Amir et al., 2000). Organizations and individuals increasingly generate and store huge amounts of data daily using IoT devices by executing different nature applications. Such applications need powerful resources for computation and storage.

Due to the limited number of available resources, proper resource allocation schemes are used to cater with all types of applications. Resource allocation (RA) scheme in the HPC environment plays a vital role in managing limited resources among multiple competing applications in a fair way that guarantees providing agreed QoS. (P. Kokkinos et al., 2009). In HPC platform a resource is any computing, storage, or communication entity that takes part in user application execution. The RA schemes consider different performance parameters based on the nature of applications. Some of the parameters are makespan minimization, computation cost

minimization, energy efficiency, bandwidth optimization, etc. These are some of the common parameters that are considered by most applications. However, some applications prefer to complete execution within specific time duration. Such applications are attached a time parameter, commonly known as an execution deadline. This type of applications are called as real-time (RT) applications. Real-time applications or services are bounded by time constraints. Such applications not merely depend on producing the correct output but the time at which the results are generated (R. L. Panigrahi et al., 2000). We use an application, service, task, and system interchangeably in this thesis. Some common examples of RT applications can be found in chemical plants, robotics, antimissile systems, pacemakers, multimedia systems, and embedded systems (J. W. S. Liu, 2000).

## 2.2 REAL-TIME APPLICATION MODEL

Consider a real-time application  $T$  as a set of multiple tasks. Assume there are  $n$  number of tasks in  $T$  and each task is characterized by the following typical parameters as shown in Figure 2.1.

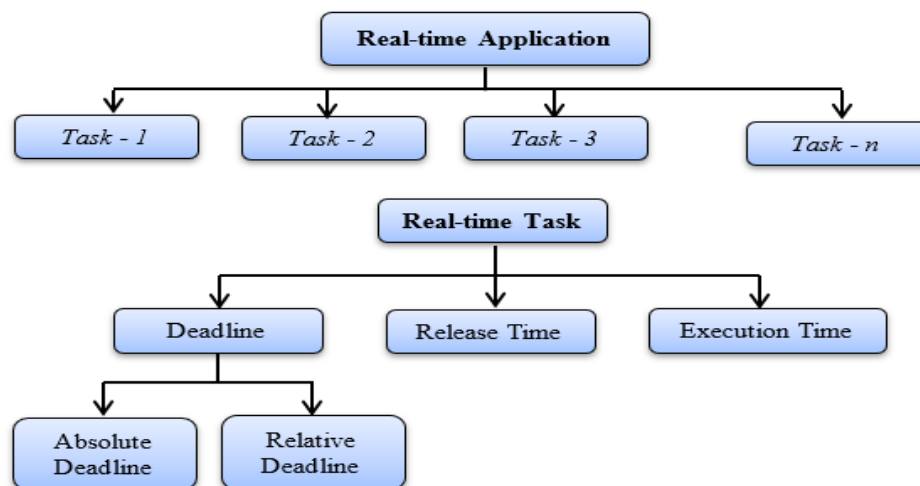


Figure 2.1 Real-time application taxonomy

1. *Deadline* – the time instant at or before which the task must complete its execution. In RT applications, mainly two types of deadlines are used: absolute and relative deadlines. The absolute deadline is a time instant which must be followed by RT task for producing timely correct results. On the contrary, relative deadline is the time duration between the absolute deadline and the release time of a task. Mathematically,

$$\text{Relative deadline} = \text{Absolute deadline} - \text{Release time}$$

2. *Release Time* – The arrival time of a task in the ready queue.
3. *Execution Time* – This is the worst-case computation time of a task for which a computational resource is required without any interrupt.
4. *Criticality* – The consequences of missing the deadline. It can be hard, soft, or firm.
5. *Response Time* – The difference between the time when a request for the resource is made and the finishing time on the resource when it is allocated.

Apart from the aforementioned parameters, the real-time tasks can be distinguished based on activation times. If the instances (jobs) of a task are activated at a constant rate, then it is called as periodic task. On the other hand, if the jobs activations of a task are not regularly interleaved, then the task is called as aperiodic task.

The focus of RA schemes for RT services is to complete execution within specified time. Based on timing constraints, RT services are broadly categorized into soft and hard RT services. In hard RT services, there is no room for missing deadline while in soft RT services, deadline missing put drastic effects on overall system performance (Bini. E & Buttazzo., 2005; Hussain et al., 2013; Qureshi et al., 2014).

### **2.3 BRIEF COMPARISON WITH THE EXISTING SURVEYS**

The existing state-of-the-art literature shows surveys on RA for different nature applications in HPC domains. But as per our exploration, no-one has consolidated RA schemes in all the emerging HPC paradigms like grid, cloud, fog, edge, and multicore systems in a single platform for special type applications i.e., real-time services. (Naha et al., 2018) surveyed RA techniques for IoT devices generating real-time data. They have considered latency parameters only and analyzed their applications in fog computing environment. They have confined their study mainly to different fog computing aspects and definitions. (W. Shu et al., 2016) provided an overview of RA in edge computing by presenting case studies ranging from smart homes to smart cities, real-time video analysis, augmented reality, healthcare monitoring, and virtual reality games. The authors (B. A. Hridita et al., 2016) surveyed mobility aware RA and scheduling algorithms. They have overviewed heuristic approaches to balance non real-time applications makespan and cloud resources monetary costs. (Mao et al., 2017) surveyed RA techniques in mobile edge computing. They have considered the communication perspective of the intended applications. They have identified limitations like high infrastructure deployment and maintenance cost, changeable human activity interaction, etc. (Mangla et al., 2016) elaborated resource scheduling and allocation schemes in a cloud computing environment. They have evaluated RA schemes based on administrative domains, virtual machine allocation and migration strategies, energy efficiency, service level agreements, and cost effectiveness. Real-time periodic systems were analyzed for schedulability on multicore systems in (Min-Allah. N et al., 2012; Nasri. M et al., 2017). The authors identified basic schedulability parameters by utilizing static priority assignment algorithms. Energy-aware RA schemes in a cloud computing environment were studied by (Beloglazov et

al., 2012). They have evaluated heuristics for efficient management of cloud data centers. The RA mechanisms for different non real-time applications on grid computing systems have been detailed by (Qureshi M. B et al., 2017). They have explained each mechanism in detail, but their study is limited to grid computing systems only. Another comprehensive survey on RA in distributed computing systems is given by (Hussain. H et al., 2013). They have studied the RA problem for mixed applications with different QoS parameters.

The following contributions highlight the novelty of this chapter.

1. The existing literature on RA shows techniques for mixed real-time and non-real-time applications on one or another distributed computing systems. There exists no comprehensive state-of-the-art comparative analysis conducted only for real-time applications. This chapter gathers and compares RA schemes only for real-time applications on HPC (grid, cloud, fog, edge, and multi-core) systems.
2. The existing research gives a plethora of RA schemes by considering certain parameters which may not give proper comparison with respect to all aspects. Instead, in this chapter, the RA schemes are compared based on the most common parameters that cover almost all of the features of all RA schemes.
3. This chapter surveys and compares real-time applications on both distributed (grid, cloud, edge, and fog computing) and non-distributed (multicore) HPC platforms, while current surveys are conducted for only one or another HPC environment.
4. This chapter evaluates and portrays each RA scheme graphically in a convinced way that is easy to understand.

5. The survey in this chapter can assist the research and development stakeholders in synthesizing and identifying their requirements for different emerging HPC paradigms encompassing different architectures and implementations. The target readers can be categorized into architecture-interested readers, algorithm-interested readers, and general-readers.

## **2.4 CHAPTER ORGANIZATION**

The chapter structure is pictorially presented in Figure 2.2. Section 1 introduces basics of the HPC systems, the real-time application model, brief comparison with the existing surveys, and overall paper organization. This section also portrays real-time application taxonomy. Section 2 evaluates resource allocation problem for real-time services. The criteria are provided for evaluating the RA schemes. Section 2 also shows a broad taxonomy of RA schemes for real-time services on HPC (grid, cloud, fog, edge, and multicore) systems. Section 3 demonstrates RA schemes for RT services in grid computing systems. Section 4 focuses on describing RA schemes in cloud computing systems. These systems are explained for accommodating real-time applications. We review the edge computing resources allocation schemes for real-time services in Section 5. Section 6 details fog computing RA mechanisms for executing real-time applications while Section 7 presents a multicore environment for such type applications. Finally, Section 8 concludes the survey.

## **2.5 HPC RESOURCE ALLOCATION PROBLEM FOR RT SERVICES**

Resource Allocation (RA) problem in HPC systems can be defined as an issue of assigning limited HPC resources in order to satisfy the performance requirements of



the competing applications according to some predefined QoS criterion such as makespan minimization, profit maximization, cost and energy efficiency, load balancing, and deadline satisfaction.

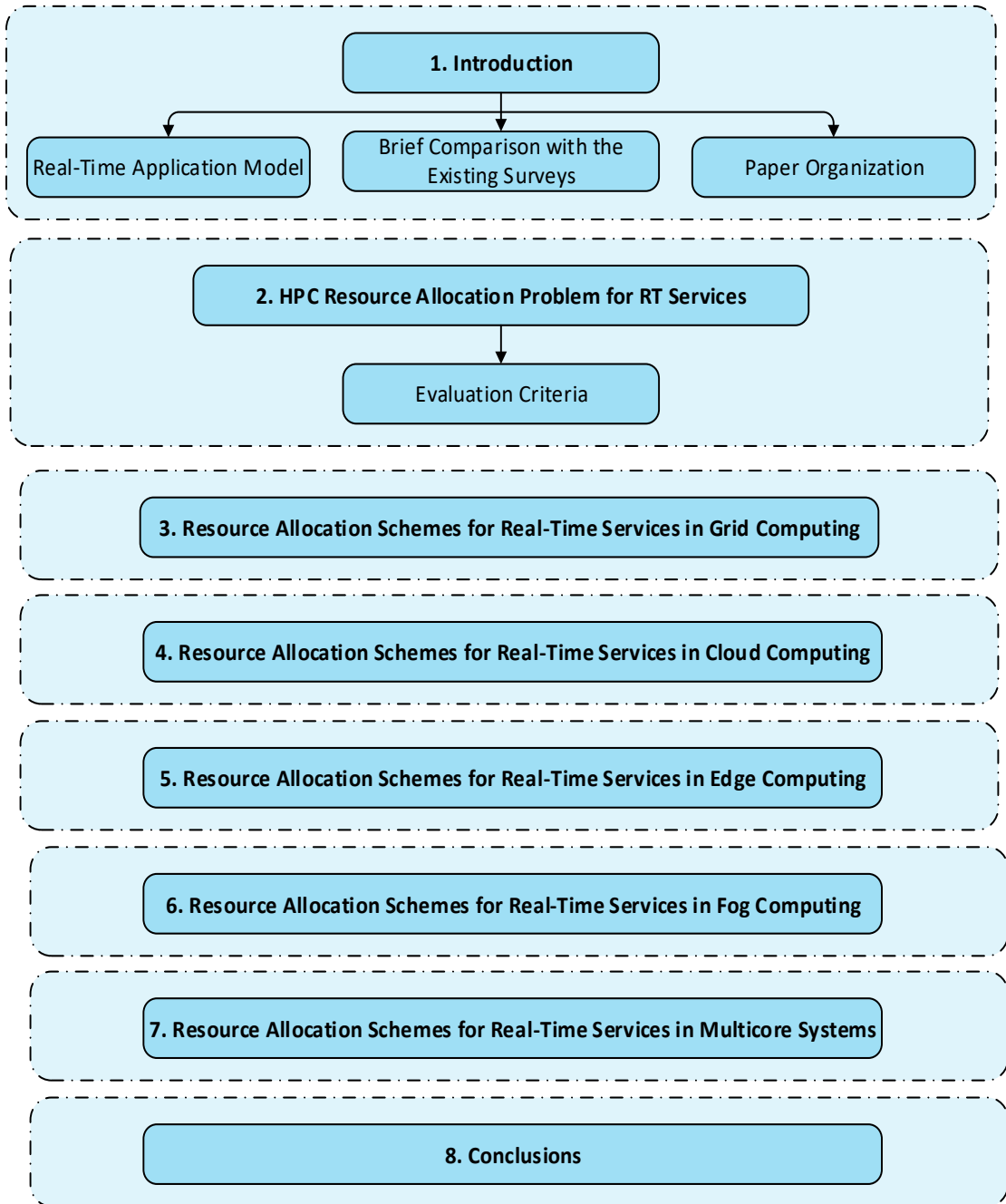


Figure 2.2 The structure of the chapter

The RT application  $T$  is composed of multiple tasks  $t_i (1 < i < n)$ , where each task is characterized by a completion deadline  $d_i$ . The  $T$  is mapped to a set of HPC resources  $J = \{j_1, j_2, \dots, j_k\}$  according to some predetermined criteria. Then the general RA problem of assigning task  $i$  to HPC resource  $j$  can be defined as  $\{t_i^j \mid \text{deadline is satisfied}\}$ .

The feasibility of the RA scheme for RT application  $T$  to HPC resources  $J$  can be defined as a function.

$$RA(T) = \begin{cases} \text{feasible,} & \text{each task deadline is fulfilled} \\ \text{infeasible,} & \text{otherwise} \end{cases}$$

The HPC resource can be defined as a machine with different capabilities. The capabilities may be processing, storage, or communication, etc. The machines with processing and communication capabilities are known as computing and network resources respectively. Some of the HPC resources have both computing and storage capabilities. These resources have associated cost of processing. The allocation cost is calculated from the resource binding process that analyses the resource performance, architecture, utilization, and processing power. Based on the architecture, HPC resources can be classified into homogeneous or heterogeneous resources. Homogeneous resources have the same, while heterogeneous resources have different designing architectures. In case of heterogeneous resources, task allocation is performed by analyzing the allocation cost. Figure 2.3 shows a broad taxonomy of RA schemes for real-time systems on the HPC platforms.

## 2.6 EVALUATION CRITERIA

The existing RA schemes for scheduling RT applications on HPC resources shown in Figure 2.3 can be compared and evaluated based on some common parameters. The

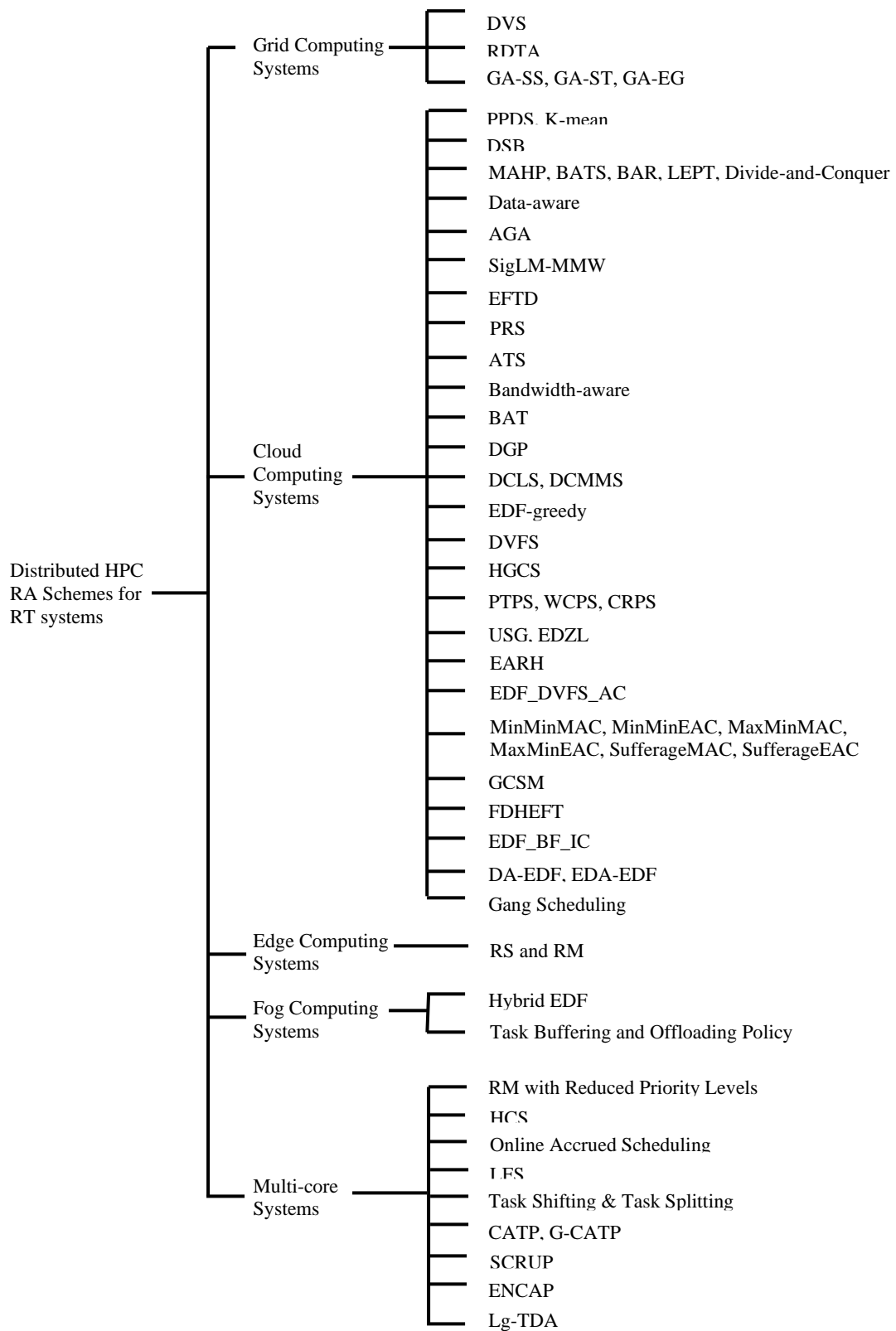


Figure 2.3 Distributed HPC RA schemes taxonomy for real-time systems

set of criteria we consider here for evaluating the work done so far include application type, operational environment, optimization goal, architecture, system size, resource type, optimality, simulation tool, comparison technique, and input data. We discuss the aforementioned aspects briefly in the following subsections and examine in detail in each RA scheme in Table 2.1.

### **Application Type**

The HPC paradigm supports various types of applications and the RA schemes are modeled based on the application type requirements. Some of the application types include real-time applications, deadlines oriented dependent applications, data-intensive real-time applications, workflow applications, etc. In this survey, we only focus on analyzing RA schemes which are developed for real-time applications. Some of the real-time applications may be data-intensive, which need data for complete processing. The dependent applications can be broadly categorized into workflow applications which consist of multiple divisible sub-applications with some sort of execution or data dependency. In case of dependency, there is communication between applications which use network bandwidth.

### **Operational Environment**

The environment consisting of executing resources on which the RA scheme is operated can be termed as operational environment. It can be categorized based on nature of resources and their operational taxonomies. For example, distributed systems, grid computing, cluster computing, cloud computing and multicore systems. These systems can be accumulated under the umbrella of HPC environment. The HPC environment can be broadly categorized into distributed HPC systems which include

grid, cloud, edge, and fog computing systems, and non-distributed systems which consist of multicore systems.

### **Optimization Goal**

Every RA scheme is designed to achieve specific metric based on which its performance is measured. For example, energy consumption, net profit gain, makespan, response time, system throughput, system utilization, applications execution within user budget constraints, etc. Some of these metrics should be maximized such as net profit gain, throughput, system utilization, application processing within user budget, while some should be minimized such as total energy consumption, makespan, and response time.

### **Architecture**

System architecture defines the nature of resources or configuration of resources within operational environment. The resources can be homogeneous or heterogeneous. Homogeneous resources have the same configuration and application execution on a single resource shows the same status of the application on all resources. Heterogeneous resources have different configurations. The diversity may be due to the resource power, short term storage capacity or instruction sets. In a heterogeneous environment, different resources give different execution statistics of applications.

### **System Size**

The system size shows the total number of resources in the operational environment. In distributed HPC environment, this number is the count of total computing and storage resource while in non-distributed HPC system, it is the total number of cores

in the system. The RA scheme considers the resource types required for the nature of executing application. If the executing application needs computing and storage resources, then the RA strategy is developed by considering the processing specifications of the resources. For example, if a system is composed of limited computing resources and a huge application needs more computing resources then the RA scheme implements proper load balancing mechanism to uniformly distribute application load on available limited resources.

### **Resource Type**

The HPC environment is composed of different types of resources like computing, storage and network resources. The computing resources only perform computation tasks. They cannot give storage facilities. The storage resources provide data storage capabilities. The network resources offer communication facilities and work for transferring data from one resource to another resource. The RA mechanism is developed by considering resource type configuration. For example, the data-intensive application engages both computing and storage resources. If these resources are located far away from one another, then the RA scheme considers computing, storage, as well as network resources for execution and data exchange among nodes.

### **Optimality**

The RA scheme that gives better results against performance metrics is called as optimal scheme. Every RA scheme strives to achieve the defined performance metric. Some of the RA schemes are near optimal or non-optimal. The near optimal RA schemes have acceptable results like optimal and very little margins exist for improving its performance, but in case of non-optimal, the RA scheme needs

revisiting. This parameter takes values like optimal, near optimal, non-optimal, or not applicable (NA) in case if it is not mentioned in the proposed scheme.

### **Simulation Tool**

The RA scheme is tested by using some virtualized environment that gives the reflection of the actual physical environment. The simulation tool consists of the same operational architecture as a real environment. Different simulation tools are used for implementing RA schemes depending upon the nature of executing applications. The results obtained by simulating application on virtualized environment have much closer accuracy of the results obtained on the actual physical environment.

### **Comparison Techniques**

This parameter reveals the validity of the proposed techniques in comparison with the other existing state-of-the-art counterparts. The results are compared, and a conclusion is drawn in the form of numerical values or graphical charts.

### **Input Data**

The Input Data attribute indicates the dataset which is considered for experimentation or simulation of a specific task scheduling and allocation scheme.

In all the above-mentioned parameters, the RT application execution within specified deadline is a common criterion for all HPC RA schemes.

Table 2.1 Comparison of the state-of-the-art HPC resource allocation schemes for real-time applications

RA Technique [Ref]	Publishing Year	Application Type	Operational Environment	Optimization Goal	Architecture	System Size	Resource Type	Optimality	Simulation Tool	Comparison Techniques	Input Data
HGCS Min-Allah et al., (2019)	2019	Real-time data-intensive tasks	Cloud computing	Makespan minimization and cost efficiency	Heterogeneous	10, 20	Computing, Storage, Bandwidth	NA	MATLAB	GA, CS	Synthetic dataset
Task Buffering and Offloading Policy Lei et al., (2019)	2019	Real-time tasks	Fog computing	Maximizing throughput and tasks completion ratio, RA balancing	Heterogeneous	Variant	Computing, Memory	NA	CloudSim	RR, MRU, AQW, FCFS, LTS, MOMIS, NMOMIS	Not mentioned
USG, EDZL Alhussian et al., (2019)	2019	Soft real-time tasks	Cloud computing Sys	Feasibility analysis, response time analysis	Heterogeneous	4 – 64 nodes	Computing	NA	Not mentioned	EDF	10000 synthetic tasksets
EDF_DVFS_AC Georgios et al., (2019)	2019	Real-time workflow applications	Cloud computing system	Cost and energy efficiency, Tasks feasibility analysis, SLA violation ratio	Heterogeneous	Not mentioned	Computing, Communication	NA	C++	EDF, EDF_DVFS	Random synthetic workload
FDHEFT Xiumin et al., (2019)	2019	Deadline-constrained workflow applications	IaaS Cloud	Cost and Makespan minimization	Heterogeneous	Not mentioned	Computing	NA	jMetal	MOHEFT, NSPSO, SPEA2, ε-Fuzzy PSO	Real-world and synthetic workflows
RS and RM Hengliang et al., (2019)	2019	Latency-critical computation-intensive applications	Cloud and Edge computing	Network delay, User QoS	Heterogeneous	1 Edge Orchestrator and 10 ESs	Computing, Communication, Storage	Yes	Hadoop 2.7.1 consisting of YARN and MapReduce,	RSH, GCP for RS and RMH, D-LAWS for RM	Dataset of SNAP
ENCAP Pavan et al., (2019)	2019	Real-time tasks	Multi-core systems	Tasks schedulability analysis	Heterogeneous	2-32 cores	Computing, Communication	NA	Feather-Trace, LITMUS	OPENMP, EDF-EN	Calandrino, Bastoni, Anderson dataset
Lg-TDA Nasro Min-Allah, (2019)	2019	Real-time systems	Multi-core system	Task schedulability analysis	Not mentioned	Not mentioned	Computing	NA	MATLAB	TDA	Synthetic dataset
PPDPS, K-mean Singh et al., (2018)	2018	Deadline-constrained dependent data-intensive tasks workflow	Cloud computing system	Deadline-constrained cost efficiency	Heterogeneous	2, 3, 4	Computing	NA	ANOVA	DPDS, IC-PCP	Cybershake, Montage, Epigenomics, Inspiral, SIPHT
DSB (Anwar & Deng, (2018)	2018	Scientific workflows	Cloud computing system	Cost efficiency, Deadline meeting	Heterogeneous	16	Computing	NA	WorkflowSim	WRPS, SCS, HEFT	Cybershake, Epigenomics, LIGO Inspiral Analysis, Montage, SIPHT
MAHP, BATS, BAR, LEPT, Divide-and-conquer Gawali & Shinde, (2018)	2018	Scientific workflows	Cloud computing system	Turnaround time, response time	Heterogeneous	20	Computing, Memory, Bandwidth	NA	Real cloud environment	BATS, IDEA	Cybershake, Epigenomics



Data-aware Nadjaran et al., (2018)	2018	Deadline-constrained bag-of-tasks applications with data requirements	Hybrid cloud computing system	User specific deadlines meeting, Execution time	Heterogeneous	19	Computing, Storage, Bandwidth	NA	Aneka	Default and Enhanced algorithms	Walkability index application
SCRUP Chaparro et al., (2018)	2018	Hard real-time systems	Multi-core system	Feasibility analysis and minimizing power dissipation	Homogeneous	4, 16	Computing, Memory	NA	Not mentioned	HTTP, TATP	3000 and 12000 synthetic datasets
Hybrid-EDF Georgios et al., (2018)	2018	Real-time IoT workflows	Fog and Cloud computing systems	Deadline miss-ratio, percent tasks executed on cloud, total momentary cost	Heterogeneous	3 fog hosts and 30 cloud hosts	Computing, Communication	NA	C++	Fog-EDF	Synthetic workload
DEFT Hui et al., (2019)	2018	Real-time applications	Cloud computing systems	Fault-tolerance, Resource utilization efficiency	Heterogeneous	Not mentioned	Computing	NA	CloudSim	NDRFT, DRFT, NWDEFT	Google tracelogs
RDTA Qureshi et al., (2017)	2017	Data-intensive real-time tasks	Grid computing system	Makespan minimization, Number of completed tasks maximization	Heterogeneous	11, 34	Computing, Storage, Bandwidth	NA	MATLAB	PTA	Synthetic dataset
AGA Mahmood & Bahlool, (2017)	2017	Hard real-time tasks	Cloud computing system	Execution time and Cost efficiency	Heterogeneous	30	Computing	NA	Wilcoxon-ranked-sum test	Greedy, GA	DAG's of varying sizes
MinMinMAC, MinMinEAC, MaxMinMAC, MaxMinEAC, SufferageMAC, SufferageEAC Stavriniades & Karatza, (2017)	2017	Real-time bag of tasks	SaaS cloud computing system	SLA violation ratio, Average result precision, Average cost per job	Heterogeneous	128	Computing	NA	C++	MinMin, MaxMin, Sufferage,	Random synthetic workload
DA-EDF, EDA-EDF Georgios et al., (2017)	2017	Real-time data-intensive BoT applications	SaaS Cloud	Impact of data locality in terms of SLA violation percentage	Heterogeneous	256 VMs	Computing, Storage	NA	C++	NDA-EDF	Synthetic workload
SigLM-MMW [18] Sangwan & Gupta, (2016)	2016	Multi-workflows	Cloud computing system	Load balancing, Average waiting time	Heterogeneous	Arbitrary	Computing	NA	CloudSim	Not mentioned	Real load traces collected on PlanetLab
GCSM Tarandeeep & Inderveer, (2016)	2016	Deadline-constrained tasks	Cloud computing system	Energy efficiency, Maximizing performance ratio	Heterogeneous	25-50	Computing	NA	Not mentioned	FCFS, SLA-based resource constraint VM scheduling, Priority-based scheduling scheme	Real-time workload traces available at Parallel Workload Archive
Gang Scheduling Approach Georgios et al., (2016)	2016	Real-time parallel applications	SaaS Cloud	Deadlines meeting, High quality results, total momentary cost, Tackling with software failures	Homogeneous	64	Computing, Communication	NA	C++	EDF, EDF_RAC, EDF_FAC, EDF_ADC, EDF_ADC_RAC, EDF_ADC_FA	Synthetic workload

										C	
RM With Reduced Priority Levels Qureshi et al., (2015)	2015	Real-time tasks	Multi-core system	Priority levels reduction, Cumulative utilization	Heterogeneous	12 cores CPU	Computing	NA	MATLAB	Traditional Approach with scheduling points	Synthetic dataset
EFTD Liu et al., (2015)	2015	DAG tasks	Cloud computing system	Normalized scheduling length (NSL) reduction, Computational cost and time minimization	Heterogeneous	10 – 130	Computing	NA	CloudSim	HEFT-AEST HEFT-TL	DAG tasks
ATS Panda et al., (2015)	2015	User requests	Multi-cloud systems	Makespan minimization, Average cloud utilization	Heterogeneous	32	Computing	NA	MATLAB	RR, CLS	Two benchmark dataset consists of 512 tasks and a Synthetic dataset
Bandwidth-aware Sindhu, (2015)	2015	Workflow	Cloud computing system	Bandwidth efficiency, Execution time	Heterogeneous	Not mentioned	Computing	NA	Eclipse	Not mentioned	Synthetic dataset
BAT Raghavan et al., (2015)	2015	Workflow applications	Cloud computing system	Execution cost minimization	Heterogeneous	3	Computing	NA	Not mentioned	BRS	Workflow consisting of 4 tasks
CATP, G-CATP Qiushi et al., (2015)	2015	Hard real-time tasks	Multi-core system	Improving system schedulability under failures	Heterogeneous	4, 8 cores	Computing	NA	Multi-core system	HAPS, BFD	Task sets generated using UniFast algorithm
EDF_BF_IC Georgios et al., (2015)	2015	Real-time workflow	PaaS and IaaS clouds	Deadlines, Cost efficiency, Execution time minimization	Heterogeneous	64	Computing	NA	C++	EDF	Real-world workflow application
PRS Chen et al., (2015)	2015	Real-time tasks	Cloud computing system	Energy efficiency, improving resource utilization, minimizing execution time	Heterogeneous	5	Computing	NA	Apache CloudStack 4.2.0, CloudSim	NMPRS, EDF, MCT, CRS	n-app, CloudSim-app
DVS Kolodziej et al., (2014)	2014	Independent batch scheduling	Grid computing system	Makespan, flow time and Energy efficiency	Heterogeneous	64, 256	Computing	NA	Sim-G-Batch Grid Simulator	GA, HGS – Sched, IGA	Kiviat Graphs, 1024 and 4096 tasks randomly generated by Gaussian distributions
DGP Deniziak et al., (2014)	2014	Soft real-time tasks	Cloud computing system	Cost optimization	Heterogeneous	6, 4	Computing, Communication	NA	Not mentioned	Not mentioned	Adaptive navigation system converted into TGs

PTPS, WCPS, CRPS Xi et al., (2014)	2014	Real-time applications	Cloud computing system	Calculating deadlines miss ratio and number of context switches	Heterogeneous	6 cores	Computing, Memory	NA	RT-Xen 2.0., Fedora 13 with para-virtualized kernel 2.6.32.25	pEDF, gEDF, pDM, gDM	Synthetic workloads
PRS Huangke et al., (2014)	2014	Real-time tasks	Cloud computing system	Energy efficiency	Heterogeneous	Infinite	Computing	NA	Apache CloudStack 4.2.0, CloudSim	NMPRS, EDF, MCT, CRS	$\pi$ -app, CloudSim-app
EARH Xiaomin et al., (2014)	2014	Real-time application	Cloud computing system	Energy efficiency and tasks feasibility analysis	Heterogeneous	Infinite	Computing	NA	CloudSim	NRHEARH, NMEARH, NRHMRARH, ProfRS, Greedy-R, Greedy-P, FCFS	Random synthetic tasks, Google cloud tracelogs
HCS Li & Yin, (2013)	2013	Flow shop scheduling problem	Distributed non-HPC systems	Makespan minimization	Heterogeneous	Not mentioned	Computing	NA	MATLAB	CS, ATPPSO, L-CDPSO, HDE, OSA, PSOMA, PSOVNS, HGA, BEST (LR), M-MMAS, QDEA	160 problems from OR library. (8 = car1 – car8 instances, 21 = rec01 – rec41 instances, 120 instances, 11= DMU instances)
Task Shifting & Task Splitting Hameed Hussain et al., (2013)	2013	Real-time tasks	Multi-core system	Load balancing	Heterogeneous	4 and 5 cores	Computing	NA	MATLAB	Not mentioned	Synthetic dataset
DCLS, DCMMS Li et al., (2012)	2012	DAG	IaaS federated cloud computing system	Resource contention, Energy efficiency	Heterogeneous	Clusters with 1024, 1152, 2048 nodes	Computing, Bandwidth, Memory	NA	Not mentioned	FCFS, FCFS (EL), DCLS (EL), DCMMS (EL)	Parallel Workload Archive (LLNL-Thunder, LLNL-Atlas, LLNL-uBGL)
GA-SS, GA-ST, GA-EG Kolodziej et al., (2012)	2012	Independent batch scheduling	Grid computing system	Makespan and energy optimization, Dynamic load balancing	Heterogeneous	64, 128, 256	DVS enabled computing	NA	HyperSim-G	Min-Min, RC, TS,	Synthetic dataset using Gaussian distributions
LFS Nasro Min-Allah et al., (2012)	2012	Real-time tasks	Multi-core system	Power efficiency, load balancing	Heterogeneous	8 and 12 cores	Computing	NA	MATLAB	FFS	Synthetic dataset
Online Accrued Scheduling Liu et al., (2011)	2011	Real-time tasks	Multi-core system	Maximizing total utility	Heterogeneous	Not mentioned	Computing	NA	Not mentioned	EDF, GUS, PP, Risk/Reward, PPOC, PPS	Randomly generated 1000 task sets
EDF-greedy Kumar et al., (2011)	2011	Real-time application	IaaS cloud computing system	Cost efficiency	Heterogeneous	Not mentioned	Computing	NA	Not mentioned	EDF, Temporal-overlap, Exhaustive search	Synthetic dataset

DVFS Kim et al., (2011)	2011	Real-time application	Cloud computing system	Power efficiency	Heterogeneous	4	Computing	NA	CloudSim	Not mentioned	Synthetic dataset
----------------------------	------	--------------------------	------------------------------	------------------	---------------	---	-----------	----	----------	---------------	-------------------

## 2.7 RESOURCE ALLOCATION SCHEMES FOR REAL-TIME SERVICES IN GRID COMPUTING

Grid computing is the integration of different hardware and the shared used of computing resources, i.e., shared infrastructure over a network for solving complex problems. In grid computing, the data is moved among different computing resources. So, managing and running distributed workflows automatically is a core feature of the grid computing.

The core essentials of grid technology are shared heterogeneous infrastructure, support of collaboration, distributed workflow management, and secure access to shared data. The general architecture of grid computing consists of a user, grid information service, resource broker, and grid resources. The user sends tasks to the grid for processing to speed up the execution of the application. The grid information service is a system that collects information of the available grid resources and send this information to the resource broker. The resource broker distributes jobs to the available grid resources based on the user's requirements for execution. The grid resources are the computing entities that execute the user jobs. The general architecture and taxonomy of grid computing environment is shown in Figure 2.4 and Figure 2.5.

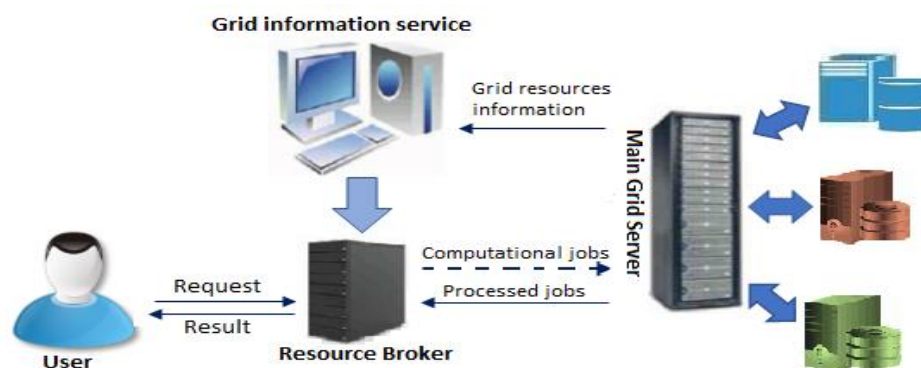


Figure 2.4 Grid computing environment

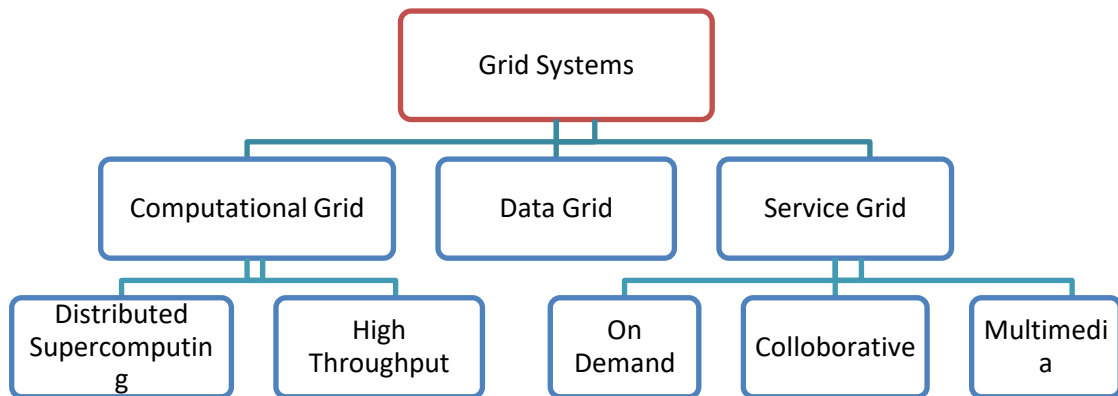


Figure 2.5 Taxonomy of grid systems

The grid computing RA schemes for real-time services are described as follows.

### 2.7.1 Dynamic Voltage Scaling

(Joanna Kołodziej et al., 2014) addressed exertion of energy spent on scheduling in grid environment, considering various grid scenarios. Grid system consists of multi-layer architecture with hierarchical management system, namely, grid fabric layer, grid core middleware, grid user layer, and grid application layer. For resource management and scheduling, two user and middleware layers are important. The three features of scheduling to define the task are static environment, batch task processing and task interrelations. The authors addressed batch scheduling in a static environment where tasks are grouped into batches and executed independently in hierarchical order. For power supply analysis, two main energy aware scheduling scenarios max-min and power supply mode are considered. The genetic algorithm is applied to solve scheduling issues by considering its six features, namely, single population in risky

mode GA(R), single population in secure mode GA(S), Multi-population in risky mode HDS-Sched(R), Multi-population in the secure mode HGS-Sched(S), multi-population island in risky mode IGA(R), and multi-population island in secure mode IGA(S). Lastly, single and multi-population are compared using empirical analysis in grid environment. The high-level scenario is shown in Figure 2.6.

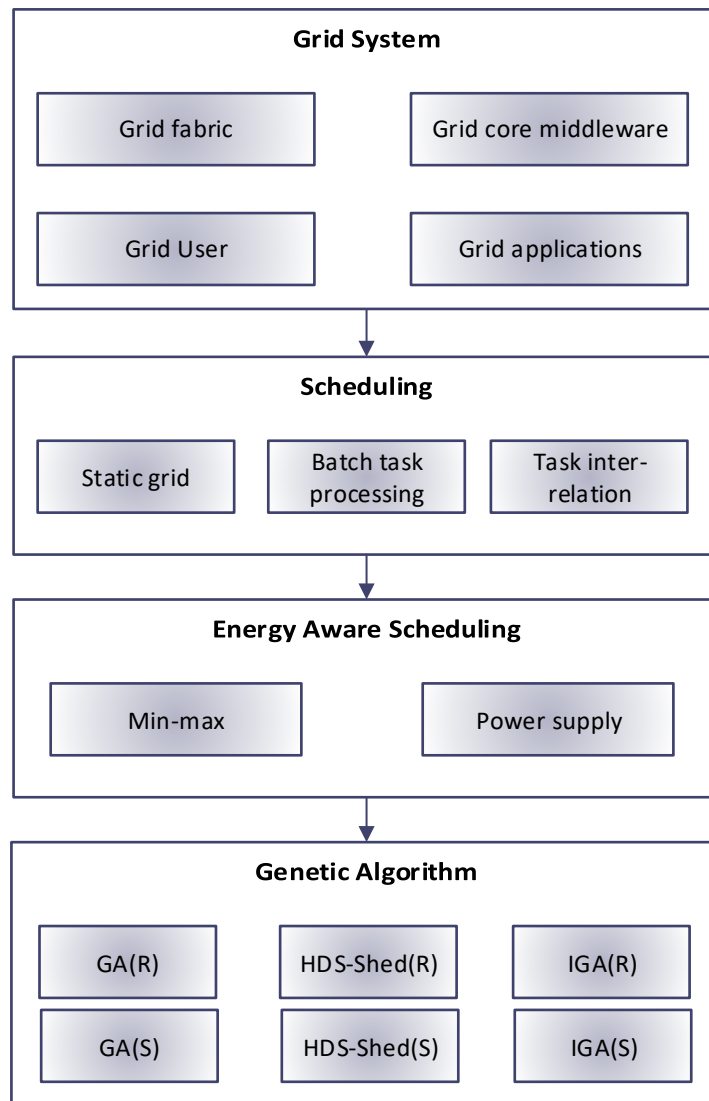


Figure 2.6 High level scenario of dynamic voltage scaling scheme in grid computing

### 2.7.2 Real-Time Data-Intensive Tasks Allocation Technique

In RDTA technique (Qureshi. M. B et al., 2017) have tested the feasibility of real-time tasks on grid computing resources. The tasks need data files for execution, which are transferred prior to or during the task processing. Initially, the basic execution demand of a task is checked on computing resources and a list of basic feasible resources is formed. Then the data file transfer time of the required files is calculated from data storage resources to the grid computing resources. After this calculation, the total execution time is calculated and if a task can be executed within its deadline by considering all the time constraints, then the task is termed as schedulable, otherwise, un-schedulable. The tasks set is schedulable only, if all the tasks in a set are schedulable. The task schedulability analysis process is portrayed in Figure 2.7.

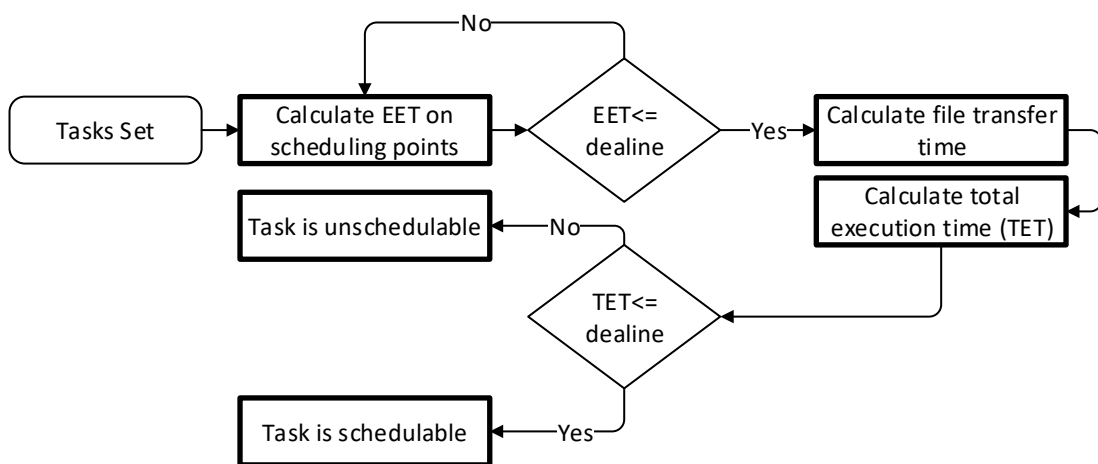


Figure 2.7 RDTA approach

### 2.7.3 Energy Efficient Genetic-based Scheduling

To achieve efficient energy in computational grids (CGs) is a core concern now a days. (Joanna. Kołodziej et al., 2012) proposed tasks in batch mode with zero dependencies between them. The authors considered two effective scheduling



functions in hierarchy mode. There are three levels of hierarchical mode which communicate over the Internet. Only one task is executed at one computational grid. No tasks are allowed to preempt the entire process. The two optimization goals used in scheduling are makespan minimization and average energy consumption. In idle mode, each machine takes minimal energy, and maximal power supply to reload the process. Three genetic methods are used to solve difficult scheduling issues. Initially, the population is generated. After selecting a parent node, crossover and mutation are applied to the individual root node and replace the parent node with the new population. Finally, an optimal individual population is generated. The proposed model outperforms than the other existing grid scheduling techniques such as relative cost, min-min, and tabu search.

## **2.8 RESOURCE ALLOCATION SCHEMES FOR REAL-TIME SERVICES IN CLOUD COMPUTING**

Cloud computing is the deployment of servers located remotely on the internet to store, manage and process the data. Such servers have large processing powers rather than a local server. Cloud computing is on-demand services delivery on a pay-as-you-go basis over the internet. There are two basic types of cloud models: service models, and deployment models. Service model refers to the kind of services the cloud offer, while deployment model refers how to deploy the application on the cloud. The service model is further sub-divided into three categories: software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS). SaaS are the cloud hosted applications and virtual desktops, PaaS are operating systems, database management and deployment tools, and IaaS are physical data centers, servers, network, virtual machines, storage and load balancers. The three types

services are provided after establishing service level agreement (SLA) between consumer and provider. The basic SLA determines the time at which the service will be provided and the corresponding usage cost (Kalaiselvi et al., 2020).

The cloud deployment models are further divided into three main categories: public, private, and hybrid clouds. Cloud computing is an automatic, pool of resources, on-demand service (pay-per-use), secure, economical, and easy maintenance system. The main benefits of cloud include its flexibility, security, accessibility, recovery from disaster, increased collaboration, document control, and automatic software adaptable system. The RA problems in cloud computing are mainly driven by monitoring, analyzing, and thoroughly checking performance of the deployed resources which ensure the agreed QoS to the intended user applications (Jyoti et al., 2020).

The general architecture of the cloud environment is presented in Figure 2.8 and a taxonomy is provided in Figure 2.9.

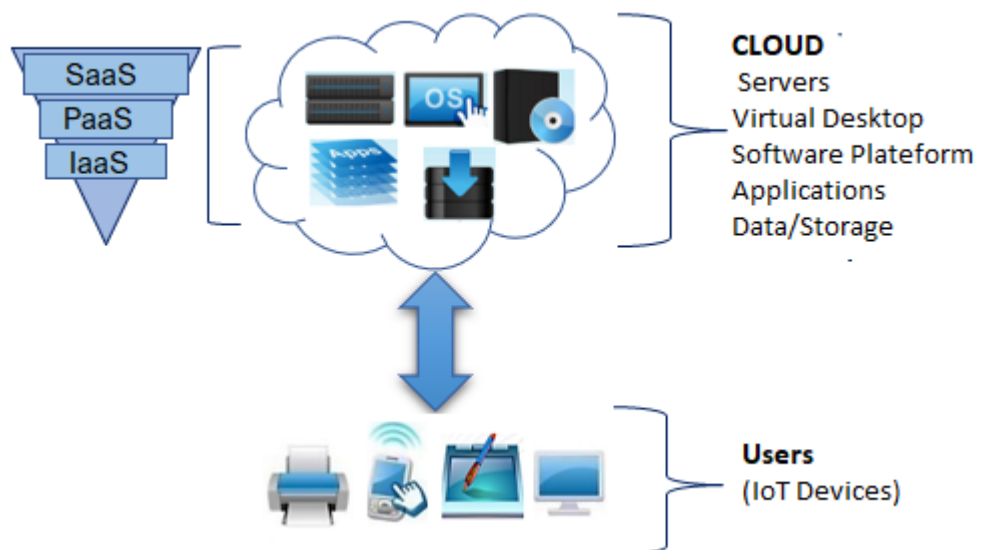


Figure 2.8 General architecture of cloud computing environment

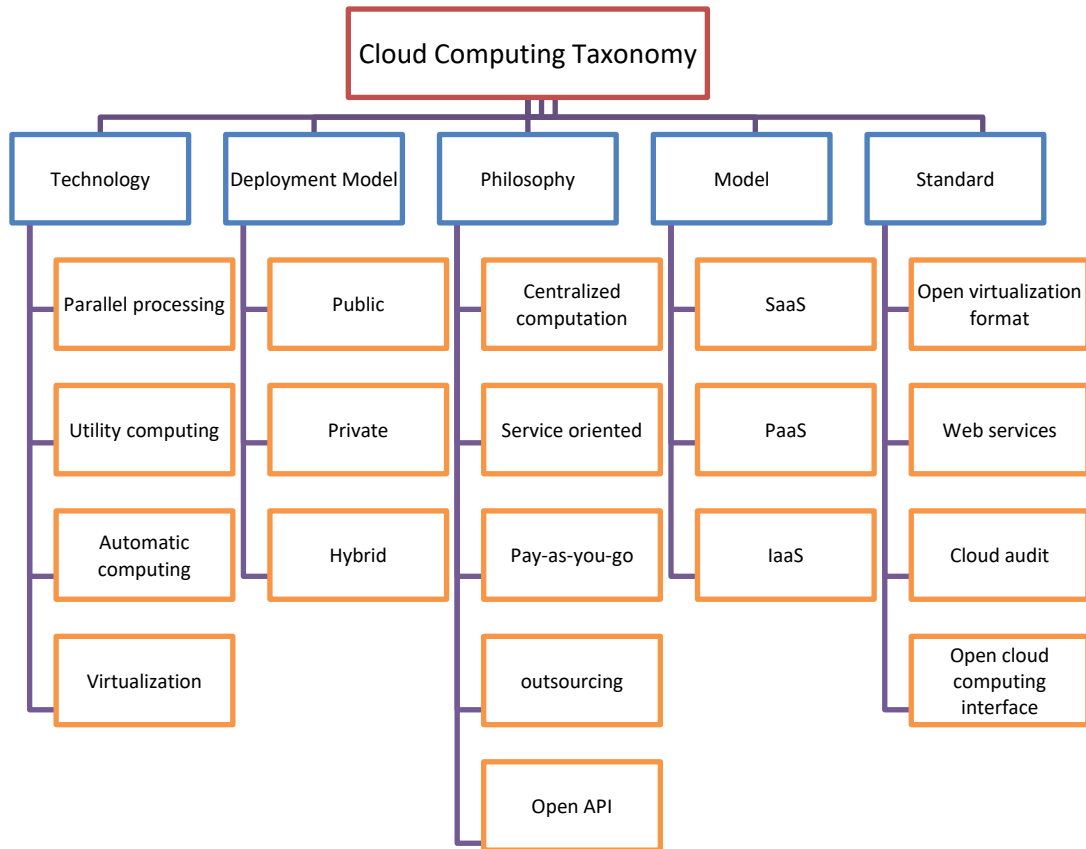


Figure 2.9 Cloud computing taxonomy

The cloud computing RA schemes for real-time services are detailed as follows.

### 2.8.1 Partition Problem-based Dynamic Provisioning and Scheduling Scheme

An innovative, cost-efficient technique known as Partition Problem based Dynamic Provisioning and Scheduling (PPDPS) for scheduling deadlines constrained workflow application was proposed by (Vishakha. Singh et al., 2018). The PPDPS algorithm works mainly in two phases, namely, Subset-Sum problem, and k-means clustering. The degree of heterogeneity of different machines is first decided and then the speed information of VMs in Millions of Instructions per Second (MIPS) is provided by the

Caching Service Provider (CSP). The k-mean algorithm is used to regulate the speed of VMs. The Subset-Sum problem approach is used to schedule the workflow to meet its deadlines at a low execution cost. It's an NP-Complete problem (Singh et al., 2018), which determines whether a set can further be divided into subsets, such that the sums of the elements in the subsets are equal. The greedy approach is used which solves the problem in polynomial time. If any of the VMs get fail during task execution, then additional storage-based model transfers the data to another VM. This model helps in easy data recovery. The PPDPS technique is compared against IaaS Cloud-Partial Critical Path (IC-PCP) (S. Abrishami et al., 2013) and Dynamic Provisioning Dynamic Scheduling (DPDS) (M. Malawski et al., 2015) techniques. The working of the proposed PPDPS model is portrayed in Figure 2.10.

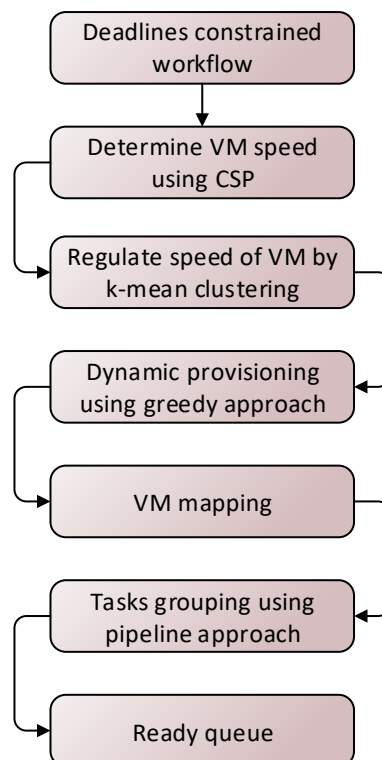


Figure 2.10 Workflow of PPDPS scheme

### **2.8.2 Dynamic Scheduling Bag of Tasks**

(Nazia. Anwar et al., 2018) proposed a Dynamic Scheduling Bag of Tasks (DSB) method for scheduling scientific workflows dynamically and elastic provisioning of VMs. The main objective of this study is to maximize the total utilization of computing resources with minimum execution cost satisfying the task deadline constraints. The proposed technique works in five steps. Initially, all tasks are assigned priorities in order to guarantee task dependencies using Heterogeneous Earliest-Finish-Time (HFT) approach. The tasks are then grouped horizontally in the same level to provide parallelism. The execution time of a task is pre-calculated that determines whether the task will fulfil the deadlines during processing. If the tasks execution exceeds deadline, the next cost-efficient VM is used to process the task. After mapping the tasks on VMs, all tasks are put into a ready queue. The task is considered ready for execution if all the precedence tasks can be completed successfully. Further, the elastic resource provisioning method is used to dynamically adjust the number of VMs instances to ensure the completion of workflow within its deadlines. It is claimed that the DSB outperforms WRPS, SCS, and HEFT techniques. Figure 2.11 shows the working flow of the DSB algorithm.

### **2.8.3 Heuristic-based Resource Allocation Schemes**

(Gawali et al., 2018) proposed heuristic methods to accomplish resource allocation and task scheduling efficiently in a cloud computing environment. Numerous size and type of data is offloaded to the cloud for execution. The Analytical Hierarchy Process (AHP) assigns a rank to each task based on priorities. Then each task is assigned to the VM using Bandwidth Aware Divisible Scheduling (BATS) and BAR optimization techniques. The load on the VM is continuously checked by using Longest Expected

Processing Time (LEPT) algorithm. If resources are unavailable, the task must wait for its turn in the waiting queue. If a resource is overloaded, the tasks are distributed on other resources using Divide and Conquer algorithm. The proposed heuristic methods improved the performance in terms of resource allocation.

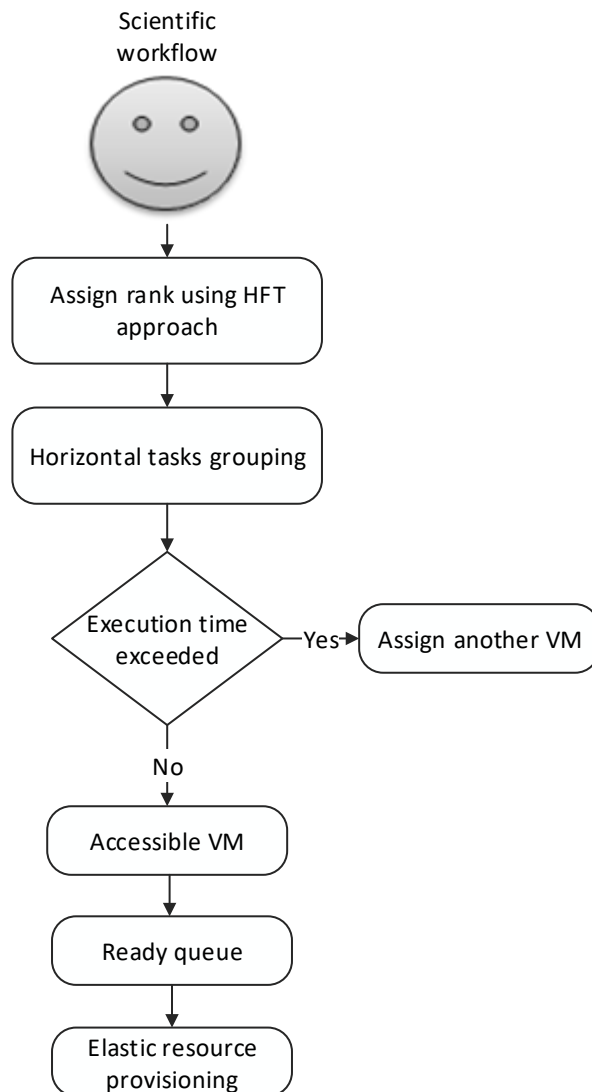


Figure 2.11 Flowchart of DSB technique

The task allocation process of the heuristic is depicted in Figure 2.12.

### 2.8.4 Data-aware Resource Allocation Scheme

To reduce the cost of execution while meeting the deadline constraints is one of the important considerable factors in hybrid clouds. A data-aware scheduling methodology is proposed by (Nadjaran. Toosi et al., 2018) to support deadline requirements of data intensive applications. Data intensive is one of the applications which are used to analyze many datasets.

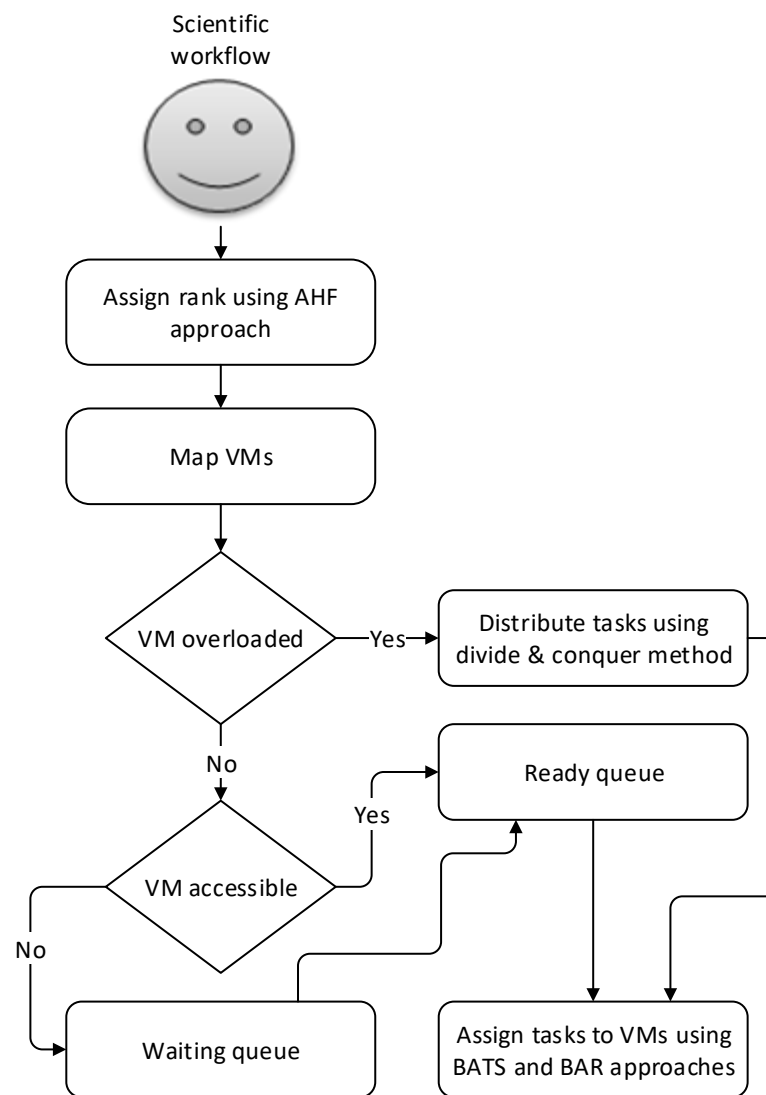


Figure 2.12 Task allocation process of heuristic-based allocation scheme

The proposed method considers the data locality, transfer time, and network bandwidth and then checks if current private cloud resources are enough to complete the task within a specific time interval. It calculates the extra resources required to execute tasks within deadlines. The remaining time in meeting the deadlines is first calculated and then the remaining number of needed resources is computed. The remaining tasks are scheduled on dynamic resources. The proposed algorithm executes data-intensive independent tasks within strict deadlines while minimizing total execution cost and the total number of required resources. Figure 2.13 shows the working of the proposed data-aware scheduling methodology.

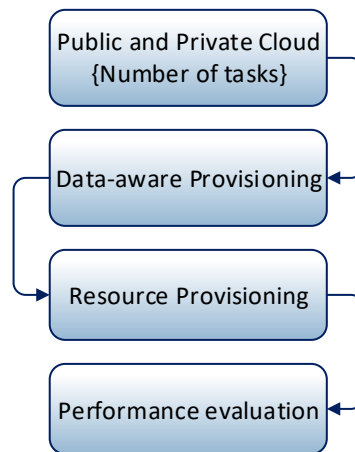


Figure 2.13 Overview of the data-aware resource allocation scheme

### 2.8.5 Earliest Finish Time Duplication Approach

Fully utilization of the selected resources is an important factor in cloud task scheduling. Delay in communication between resources degrades overall system performance. To overcome these limitations, (Liu, Z et al., 2015) proposed Earliest Finish Time Duplication (EFTD) approach to pre-process the cloud resources effectively. Directed Acyclic Graph (DAG) method is used to improve the utilization



of resources. In this method, tasks with the highest priority are assigned to the selected resource for further processing. Three steps are performed to assign resources to specific tasks: calculating Earliest Start Time (EST), Earliest Finish Time (EFT), and task allocation. In this approach, initially EST is calculated to regulate processing unit and an EST value is assigned to processing resource. Then the EFT is calculated to choose processing unit and assign it smallest EFT value. Finally, the results of both times (EST and EFT) from ready tasks are compared. The high priority task is assigned to that resource whose both times are same. The next task with lower priority which has both times different is chosen from the ready queue.

To reduce the processor scheduling time, parent nodes of two key tasks are duplicated. In this way, the communication process is improved as compared to the other counterparts (HEFT, AEST, HEFT-TL). The whole scenario is depicted in Figure 2.14.

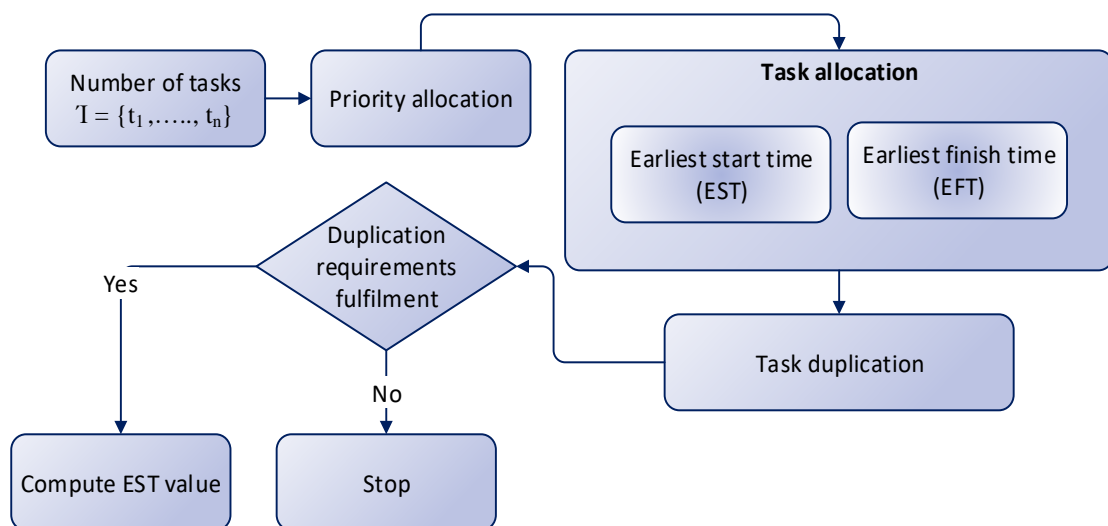


Figure 2.14 Workflow of EFTD RA scheme

### **2.8.6 Task Scheduling and Load Balancing Technique**

Load balancing is an important factor while dealing with numeral requests sent to the server through a network. (Sangwan. A et al., 2016) proposed task scheduling algorithm in cloud computing based on load balancing to meet user demands and make full use of resources. The whole process considers the above mentioned two aspects in four phases. Initially, when new workflow arrives, it is submitted to the pre-processor component for computing different attributes of all ready tasks. Then the ready task is placed in a ready queue for further processing. When services become available, the scheduler executes all tasks available in the ready queue and when the task completes its execution, the executor notifies the pre-processor of the completed task. This technique tries to balance the load on all available resources so that tasks can be completed in prespecified deadlines. The whole process is briefly summarized in Figure 2.15.

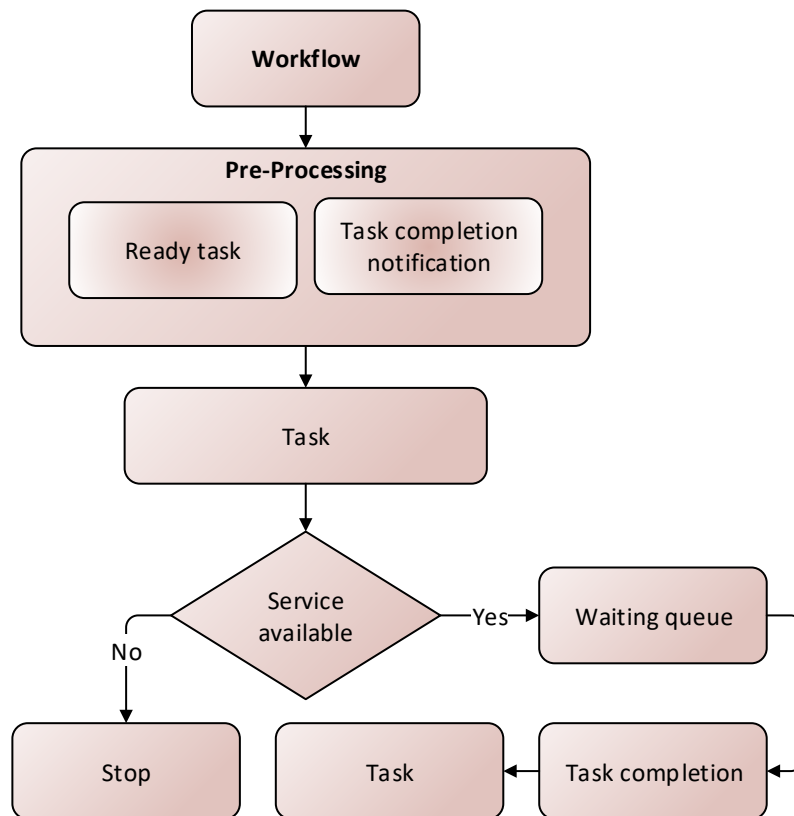


Figure 2.15 Task scheduling and load balancing technique

### 2.8.7 Proactive and Reactive Scheme

To meet real time cloud computing environment, there was a need to reduce system's energy consumption. To resolve this issue, the authors (Chen. H et al., 2015) proposed PRS (Proactive and Reactive Scheduling) procedure to make efficient utilization of system resources with reduced energy consumption. When a new task is arrived in the system, the PRS algorithm checks whether the task has urgency. If it has urgency parameter, then it is referred to the urgent task queue. Otherwise, task is further sent to waiting queue in ascending order by their laxity. Additionally, PRS checks the requirements of waiting task not to exceed the available system resources. Later, tasks are scheduled to virtual machines for execution. The whole scenario is illustrated in Figure 2.16.

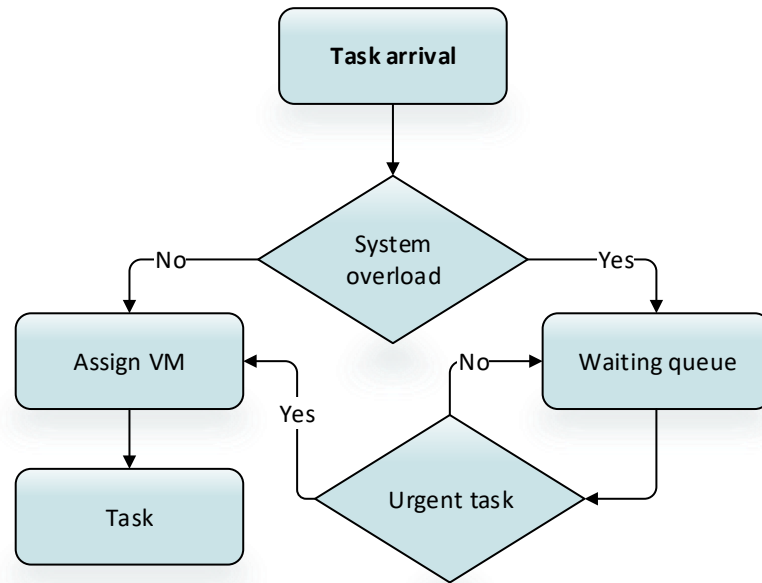


Figure 2.16 Working of PRS RA scheme

### 2.8.8 Allocation-aware Task Scheduling

Allocation aware Task Scheduling (ATS) algorithm is proposed by Sanjaya K. Panda for multi-cloud environment (Panda. S. K et al., 2015). Each cloud consists of different datasets to deploy virtual machines (VMs). The proposed model comprises of three steps, i.e., matching, allocating, and scheduling. In matching phase, cloud manager preserves a universal queue to insert an incoming request from users. The requests are served as First in First Out (FIFO). The manager matches the task with other virtual machines to find optimized VM among all. Then, the request is removed and the completion time of task on a specific virtual machine is computed. Thus, one task is selected from the global queue at a time. Additionally, the virtual machine is found by manager to grasp minimum completion time for task supervised by cloud. Task scheduling is decided in allocation phase. In allocation phase, resources are assigned to tasks. The scheduler executes all tasks to carry out the computation. Each cloud resource executes more than one task concurrently. The experimental results

demonstrate better performance than Round Robin (RR) and Cloud List (CLS) scheduling algorithms. The entire process is portrayed in Figure 2.17.

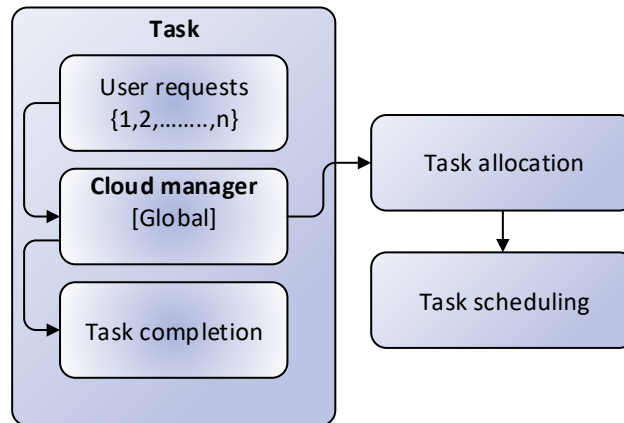


Figure 2.17 High level view of ATS algorithm

### 2.8.9 Bandwidth-aware Resource Allocation

Quality of Service (QoS) attainment is incredible essential of all the users. In cloud computing, processing a task in a flow is vital. A non-linear programming model for task scheduling is proposed by Sindhu, (2015). In this model, several tasks are ready to be scheduled. Various resources are assigned to each task. Among all, some resources are used, and the remaining are being idle. To avoid resource power wastage, limited network bandwidth is considered. Based on accessible bandwidth, each task is forwarded to every VM. The proposed model produced better accuracy than other existing models by taking full utilization of resources and reducing the waiting time. The working flow of this technique is shown in Figure 2.18.

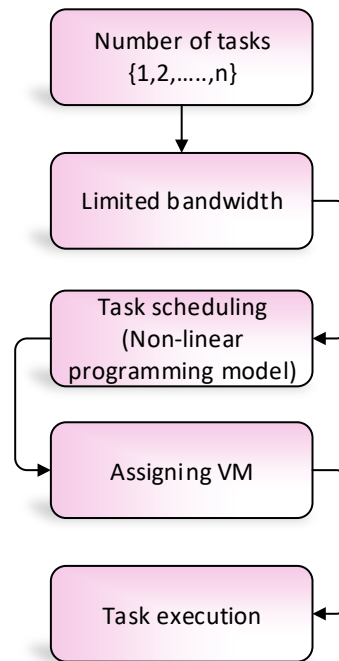


Figure 2.18 Bandwidth-aware RA scheme workflow

### 2.8.10 Bat Approach for Resource Allocation

(S. Raghavan et al., 2015) proposed Bat approach for the optimum solution of workflow scheduling. The workflow scheduling contains two chunks, namely, task scheduling and mapping tasks and resources. In this approach, task and resources are mapped together to reduce the entire cost of execution. The model contains three resources and four tasks, each resource has different execution costs. Furthermore, the cost of each task is computed on each resource. With the help of mapping function, a list is maintained when every task is mapped with resource based on minimal value. Each resource doesn't contain more than one task. Finally, the result is computed and compared to the Best Resource Selection (BRS) algorithm. The model showed overall nominal cost. The working of BAT algorithm is shown in Figure 2.19.

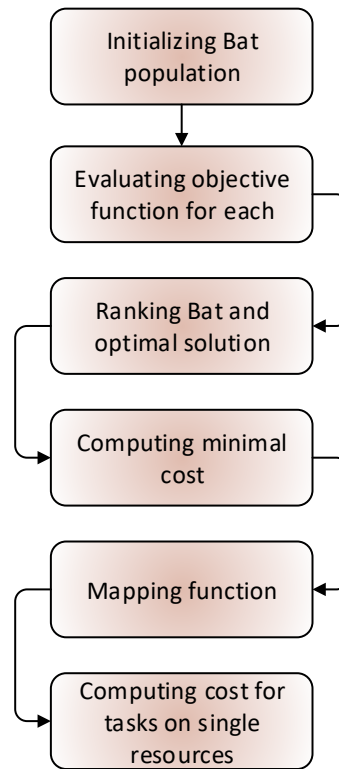


Figure 2.19 Working of BAT algorithm

### 2.8.11 Developmental Genetic Programming

In real-time computing, practicing cloud infrastructure is a new concept. Resources are assigned to the tasks through cloud infrastructure. It is very important to design an efficient and effective procedure for resource allocation through cloud. (Slawomir. Bak et al., 2014) analyzed the problem of cloud resource allocation to minimize execution cost in real-time applications. The existing methodologies i.e., Iterative Improvement (Panda. S. K et al., 2015) and Constrained Logic Programming (Sindhu, 2015), briefly explained and resolved the application cost problem, but still there is a chance of improvement to minimize the execution cost. An efficient genetic procedure is proposed to minimize the cost of applications with high QoS in a cloud computing environment. Initially, the system comprises of distributed methods considering the worst-case scenario where all tasks are started at equal time. All tasks are scheduled in

a fixed order with a definite time frame. In addition, the methods are converted into several task graphs due to static scheduling. Also, primary population of genotypes is created and solved through genetic procedure. Here, the developmental scheduling algorithm is used for assigning and scheduling all tasks which outperform the ideal solution. The whole process is shown in Figure 2.20.

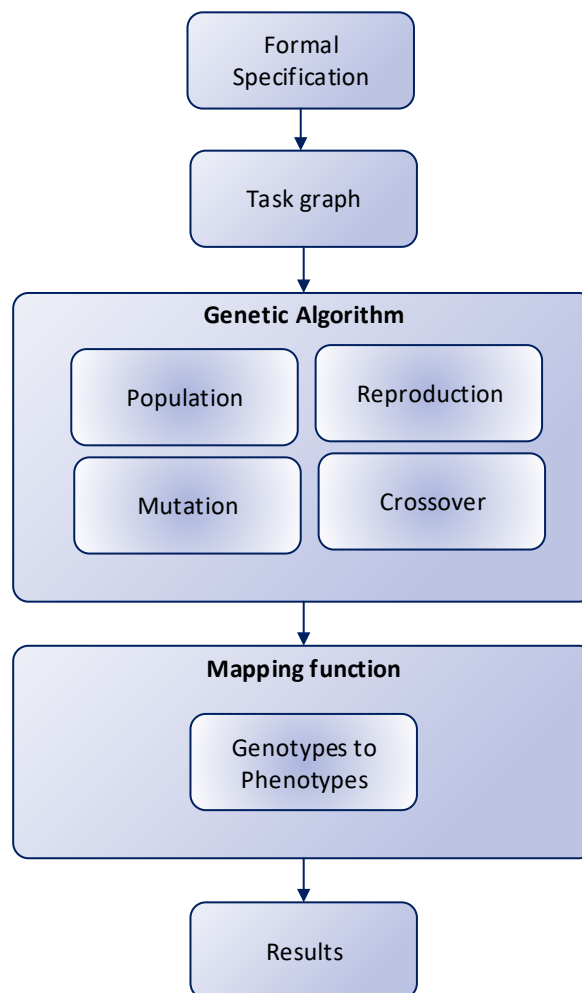


Figure 2.20 Flowchart of Developmental Genetic algorithm

### 2.8.12 Federation-based Resource Allocation Scheme

(Jiayin. Li et al., 2012) proposed a new approach in a cloud computing environment online for task scheduling on Infrastructure-as-a-Service (IAAS) cloud. In this



research, various data centers meet with the help of federated method. Each data center contains manager server which holds information about VMs and communicate with each other respectively. The tasks are stored in a database in the cloud, then cloud collects tasks and distribute it in the entire cloud system considering resource accessibility in the other clouds. The resource scheduling is being checked by monitory infrastructure. The infrastructure, producer, and consumer are communicated with each other in two modes, namely, push and pull mode. While making decisions, consumer pulls information about resources from other existing clouds. After finishing task execution, producer pushes the data to the consumer. Two distinct methods, namely, Advance Reservation (AR) and Best Effort (BE) are used for hiring capacities from cloud computing infrastructure. Both push and pull modes are used in the basic resource allocation model. Finally, Directed Acyclic Graph (DAG) is used in the application model. Energy aware local mapping significantly reduces energy consumption in the federated cloud computing environment. This procedure is illustrated in Figure 2.21.

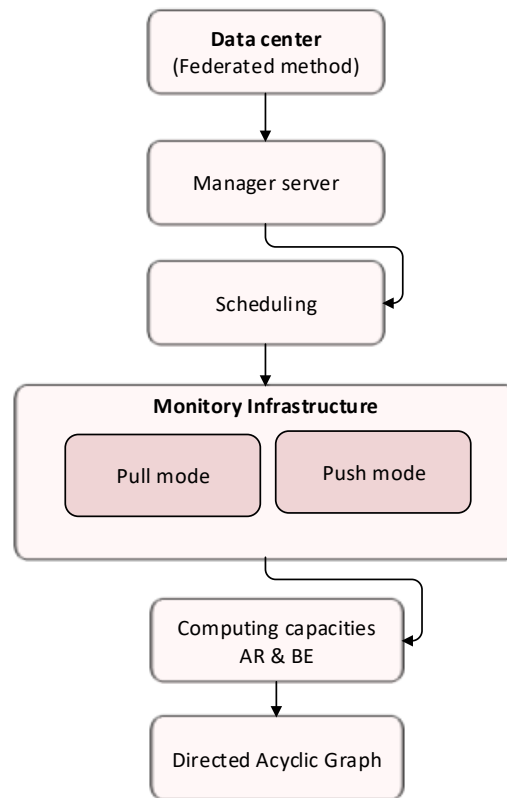


Figure 2.21 Federation-based RA scheme

### 2.8.13 Adaptive Genetic Approach

Allocation and scheduling of tasks on a VM is a difficult job in a hard-real-time environment. To resolve this problem, (Amjad. Mahmood et al., 2017) proposed Adaptive Genetic Algorithm (AGA) in a cloud computing environment. The model consists of numerous tasks having the work pressure to complete its tasks while meeting the deadline. Tasks have a precedence relationship when communicate with each other on VM denoted by a Directed Acyclic Graph (DAG). The scheduling of tasks is done using chromosomes to generate the population. Multiple crossover is performed on randomly selected chromosomes without troubling the scheduler. Furthermore, mutation is accomplished by selecting a gene from chromosome. Each gene is capable of changing bits with some probability. The proposed AGA model is compared with the other techniques such as DAG technique, greedy search approach

and non-adaptive GA. Better outcomes are achieved by the proposed algorithm. This process is shown in Figure 2.22.

#### **2.8.14 Dynamic Fault-tolerant Elastic Scheduling (DFES)**

It is vital to improve the resource utilization in cloud computing environment. (Hui. Yana et al., 2019) proposed a fault tolerant algorithm to accomplish resource utilization. The data center comprises of multiple hosts, each having its own VM. A user maintains task flow in a task queue. The system schedules the task which keeps record of both task scheduler and performance monitor. The monitor is responsible to retain the status of the system performance. On the other hand, task scheduler schedules task when getting feedback from monitor. The task scheduler, data center, and monitor communicate with each other by using star topology. The performance of the model is verified by using CloudSim simulator through Google tracelogs. Figure 2.23 shows the whole process of DFES algorithm.

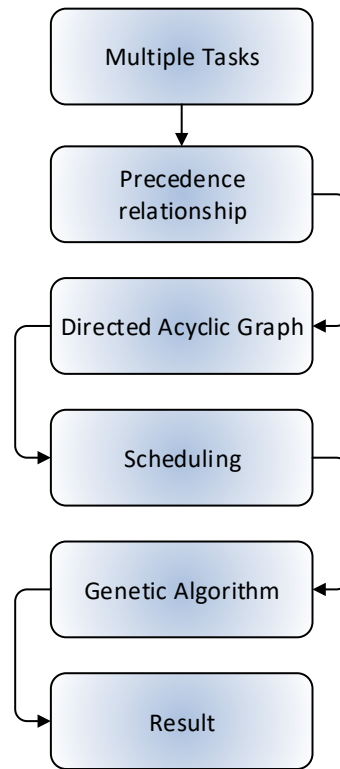


Figure 2.22 Simplified flowchart of AGA scheme

### 2.8.15 Earliest Deadline First Greedy Approach

(Karthik. Kumar et al., 2011) proposed a VM allocation procedure for real-time tasks. The main objective of the proposed technique is to decrease the cost of tasks allocation while meeting deadlines. The authors proposed a greedy approach based on Earliest Deadline First (EDF) algorithm. Initially, tasks are allocated separately to access VMs. When VMs became inadequate to accomplish the task on time, then based on lookup table, other low-priced VMs are selected by greedy approach to complete the tasks before expiry of the time constraint. When tasks overlap each other and produce high cost, then identifying overlapping tasks and allocating resources together create outcomes in polynomial time. The proposed model shows the allocation of tasks. This process is shown in Figure 2.24.

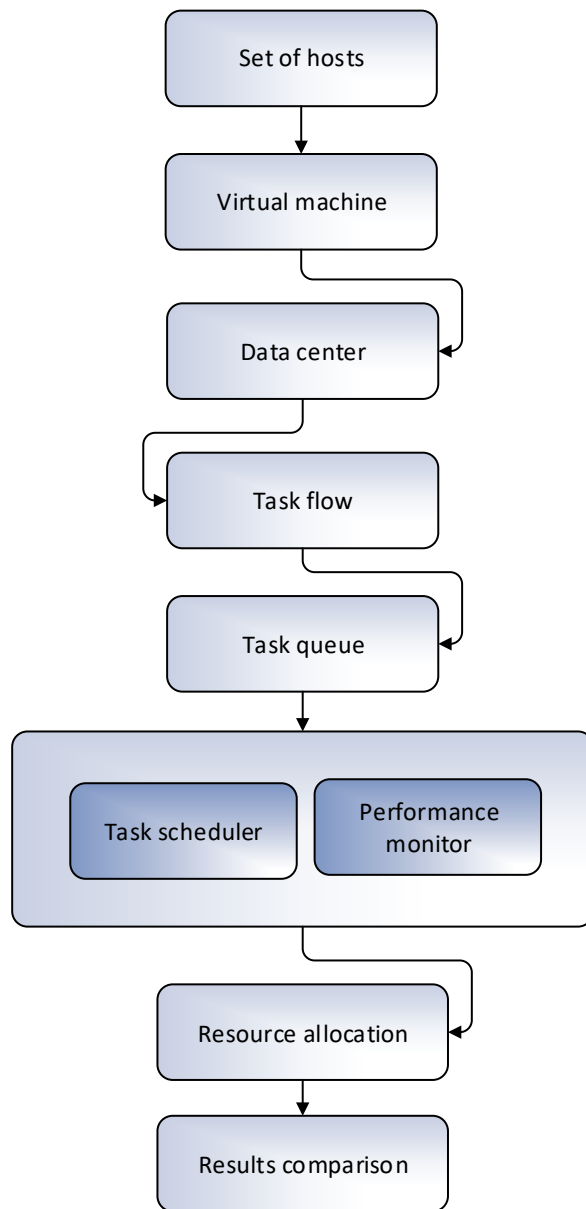


Figure 2.23 Workflow of DFES algorithm

### 2.8.16 Deadline-oriented VM Allocation Approach

(Kyong. Kim et al., 2011) proposed an algorithm not only to reduce the task allocation cost but, also increase the system consistency for both hard and soft real-time services. In hard real-time service, if a task does not meet the timing constraint, then penalty is charged. In soft real-time service, if a task does not meet time constraints, then there is

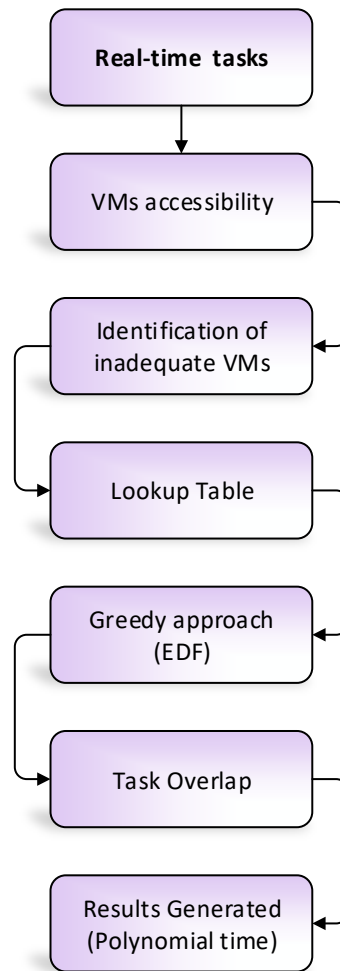


Figure 2.24 EDF Greedy task allocation procedure

no penalty and results are accepted, but the Quality of Service (QoS) is decreased. Depending on the deadline type, the user requests hard or soft VM for further processing. Initially, the user requests a VM by providing all the information to the vendor about real-time applications. Then the provided information is first analyzed for creating one real-time VM (RT-VM) request and the vendor offers a VM from the cloud computing environment. The VM meets the task's requirement constraint and its information is submitted to the user later. Finally, the user executes the real-time applications. The whole scenario is depicted in Figure 2.25.

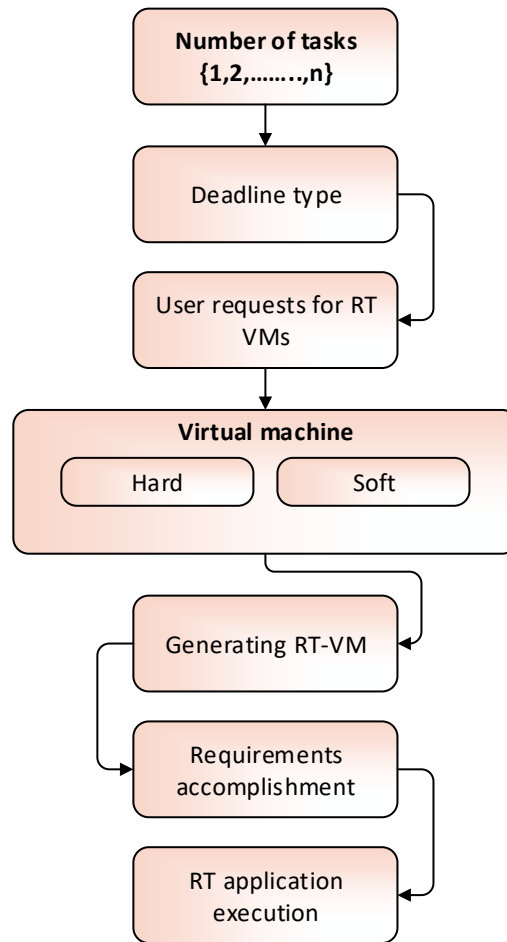


Figure 2.25 Deadline-oriented VM allocation approach

### 2.8.17 Approximate Computation-based Resource Allocation Scheme

In a cloud computing environment, QoS provision is preferable requirement for the consumer. (Georgios L. Stavrinos et al., 2019) proposed approximate computation-based scheduling technique for real-time system. To schedule tasks in a ready queue, heuristic is applied to select task and VM. In the beginning, tasks are prioritized based on EDF policy. In case of tie, task with the highest cost is selected. Once the task is selected by the scheduler, it is assigned to a virtual machine based on the Earliest Estimated Finish Time (EEFT) approach. The proposed model is made energy efficient through the utilization of DVFS technique during the VM selection process. Also, the model is cost effective when it reduces the idle time of the VM. The

proposed model provides better results than other existing techniques considering total energy consumption, SLA violation ratio, average result precision, and total monetary cost as performance parameters. The working of the proposed model is represented in Figure 2.26.

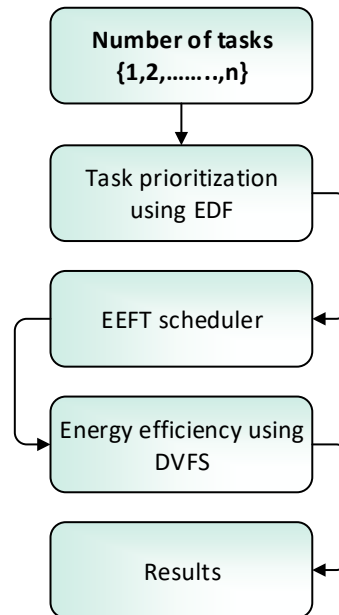


Figure 2.26 Flow of approximate computation-based RA scheme

### 2.8.18 Periodic Server Scheduling Scheme

Numerous systems share single computing platform to enrich flexibility and reduce the execution cost through virtualization equipment instead of using separate hosts. To overcome this issue, (Xi. Sisu, 2014) proposed novel scheduling structure in real-time environment. All the incoming tasks are prioritized considering pre-emptive fixed scheduling. Then, these tasks are scheduled according on the virtual processing units. Every operating system is liable on tasks scheduling. The processing is executed in two conditions: runnable and non-runnable. Xen scheduler schedules dependent functions on distinct VM and spread its other parts. The jobs continued to execute on a



single core hierarchy considering soft real-time tasks. Also, tasks are implemented on sporadic server where do-schedule method is called when virtual processing differs from running tasks. The proposed model performance is computed on different quanta whose range lies in 1ms – 10ms. The proposed model achieved better results at fine-grained quantum considering tasks overhead.

### **2.8.19 Heuristic-based Earliest Deadline First Scheme**

In software services, the data locality problem exists nowadays. There is a need to reduce the problem of workload during execution of multiple tasks. (Georgios. L et al., 2017) presented a solution to overcome heavy workload problem. In this scenario, tasks are dynamically scheduled in cloud computing under different circumstances. Three heuristics are applied on a real-time task set such as Non-Data-aware Earliest Deadline First (ND-EDF), Data-aware Earliest Deadline First (DA-EDF), and Earliest Data-aware Earliest Deadline First (EDA-EDF). In ND-EDF, tasks are entered in global queue for central scheduling. Then tasks are sorted in descending order of mean computational costs. Tasks having largest cost are selected first for scheduling. After scheduling, tasks are assigned to the VM with the earliest start time. The VM executes tasks according to the earliest deadline. In DA-EDF, tasks are scheduled like NDA-EDF via the largest mean computational costs. The tasks are assigned to the VM having earliest start time. But the tasks are executed with their job's deadline. In EDA-EDF, the procedure of tasks selection is same as other two procedures. Here, tasks are executed according to the earliest start time employing EDF policy. NDA is matched with the other two heuristics DA and EDA-EDF to aware the system performance on data locality. In the proposed model, data locality is considered during scheduling.

### **2.8.20 Application-directed Check Pointing and Approximate Computation Scheme**

(Georgios. L et al., 2017) focused on reducing cost and fault tolerance in SaaS cloud computing environment to achieve good quality within time constraints. The six scheduling techniques Earliest Deadline First (EDF), Earliest Dead-line First with Restricted Approximate Computations (EDF-RAC), Earliest Deadline First with Full Approximate Computations (EDF-FAC), Earliest Deadline First with Application-directed Checkpoints (EDF-ADC), Earliest Deadline First with Application-directed Checkpoints and Restricted Approximate Computations (EDF-ADC-RAC), and Earliest Deadline First with Application-directed Checkpoints and Full Approximate Computations (EDF-ADC-FAC) are compared to compute the performance of overall system. In EDF, tasks are in queue according to earliest Estimated Start Time (EST). The tasks are engaged in selected queue considering EDF. In EDF-RAC, tasks in queue are allocated through EDF. In EDF-FAC, tasks selection is same as EDF and EDF-RAC procedures. The tasks are scheduled only to execute the mandatory portion. In EDF-ADC, tasks are employed according to directed check pointing technique. In EDF-ADC-RAC, tasks are employed considering both applications directed check pointing and approximate computations. In EDF-ADC-FAC, tasks are scheduled to execute mandatory portions. The comparison among scheduling algorithm proved that EDF-FAC produced better results and EDF-ADC gave the best results in worst scenarios.

### **2.8.21 Earliest Deadline First and Unfair Semi-Greedy Approach**

(H. Alhussian et al., 2019) proposed cloud computing architecture that consists of two parts: Master Node (MN) and Virtual Machine (VM). In this scenario, it is assumed

that MN has VMs pool. When real-time job comes to the MN, the master scheduler picks a VM resource from the pool and assign tasks to it until VM is fully loaded. When the minimum required number of VM is determined, the MN scheduler assigns real-time services to them. Each VM node has deadline look-a-head module that gives information regarding deadline-miss event. In case of such event, the event is removed from scheduling queue and put into urgent queue. The MN scheduler at that point looks for a VM node with an unoccupied processor and assigns the urgent tasks to any such node that is found unoccupied. Sometimes, the MN scheduler also makes another VM node and appoints the urgent tasks to it. When VM deadline is suspected, it removes those tasks from waiting queue and sends the suspected tasks to the MN. By following this procedure, tasks are expected to meet the deadlines.

#### **2.8.22 Dynamic Proactive Reactive Scheduling**

(H. Chena et al., 2014) proposed Proactive and Reactive Scheduling (PRS) algorithm that dynamically exploits PRS methods for scheduling real-time tasks. The tasks are assumed aperiodic and independent. The authors examined how to reduce the system's energy consumption while guaranteeing the real-time constraints for green cloud computing where uncertainty of task execution exists. The proposed PRS algorithm achieved improved performance as compared to four typical baseline scheduling algorithms: non-Migration-PRS (NMPRS), Earliest Deadline First (EDF), Minimum Completion Time (MCT), and Complete Rescheduling (CRS).

#### **2.8.23 Energy-aware Resource Allocation**

Energy conservation is a big issue in cloud computing systems. If it is handled properly, then reducing operating costs, system reliability, and environmental

protection factors can be achieved effectively. Existing power-aware scheduling approaches provide promising way to achieve this goal, but the issue is that they are not real-time tasks oriented and thus lacking the ability of guaranteeing system schedulability in such situations. To solve the aforementioned problem, (X. Zhu et al., 2014) proposed rolling-horizon scheduling architecture for real-time tasks scheduling in virtualized clouds and energy-aware scheduling algorithm for real-time tasks. The tasks are aperiodic and independent. Two other strategies “scaling up” and “scaling down” are proposed for making trade-offs between task’s schedulability and energy conservation.

#### **2.8.24 Bag of Tasks Scheduling with Approximate Computation**

The authors (G. Stavrinides et al., 2017) devised six different techniques, that are MinMinMAC, MinMinEAC, MaxMinMAC, MaxMinEAC, SufferageMAC, and SufferageEAC for scheduling batch of real-time tasks (BoT) on SaaS cloud computing systems. All tasks are independent and non-preemptive which avoid performance degradation. Each task is assigned some weight which shows its total number of required computations. If a task fails to finish on time, it takes non-completed status and gets out of the scheduling queue because its deadline is non extendable. A task gets completed status when all its mandatory parts (jobs/subtasks) are executed on time. The completion of optional subtasks may refine the results. In the proposed model, a task can get a finished, partially finished, or skipped status. If a task is partially finished, it is called as approximate and the output may get affected. In any case other than approximate, a job is lost.

In the proposed model, when a task arrives at the central scheduler, it enters the queue and scheduler is invoked. The load on VM is calculated and the following steps are performed.

1. Estimate the completion time of a task.
2. Calculate idle time of a processor.
3. Arrange queue according to the EDF algorithm.
4. Calculate minimum completion time (MCT) of a task.

The detailed working of the proposed framework is depicted in Figure 2.27.

#### **2.8.25 Green Cloud Scheduling Approach**

(T. Kaur et al., 2016) proposed a technique to efficiently utilize the nodes in a cloud computing system to save energy consumed during the process. With the help of virtualization, heterogeneous types of tasks are efficiently assigned to the nodes within their deadline limits that are energy efficient using Green Cloud Scheduling Model (GCSM). Running tasks, resources utilization, and energy statistics information are stored in a database. Cloud user submits the task(s) and provides the deadline information for the task(s). The tasks handler and analyzer section check whether the current task along with the constraints provided by the user exists in the database. If it is there, the task is assigned the required resource. If the task is not found in the current database, then a special unit called Green Cloud Scheduler (GCS) efficiently perform scheduling of tasks on appropriate nodes within the deadline limits of the tasks. The GCS also tries to eliminate idle node(s) within the system, and thus minimizes the energy consumed by unnecessary nodes. The proposed system saves energy up to 71%, while 82% of the tasks are completed within their deadline constraints.

### **2.8.26 Fuzzy Dominance Sort-based Resource Allocation Technique**

(X. Zhou et al., 2019) minimized cost and makespan simultaneously for workflows deployed and hosted on IaaS clouds. This scheme proposes a Fuzzy Dominance sort-based Heterogeneous Earliest Finish Time (FDHEFT) algorithm which closely integrates the fuzzy dominance sort mechanism by using Heterogeneous Earliest Finish Time (HEFT) list scheduling heuristic. The proposed scheme achieved significantly better cost-makespan tradeoff fronts with remarkably higher hyper volume and can run up to hundreds of times faster than the state-of-the-art algorithms. Two sets of simulation experiments are implemented on real-world workflows and synthetic applications to validate the effectiveness of the proposed FDHEFT. The experiments are based on the actual pricing and resource parameters of Amazon EC2. The produced results show supremacy of the FDHEFT approach with respect to cost-makespan trade-offs and a lower CPU runtime when compared to the other peer approaches like  $\epsilon$ -Fuzzy PSO, NSPSO, SPEA2, and MOHEFT.

### **2.8.27 Best Fit with Imprecise Computations**

(G. Stavrinides et al., 2015) proposed variant of EDF scheduling scheme called EDF-BF-IC for the real-time workflow applications scheduling in heterogeneous Platform-as-a-Service (PaaS) cloud that combines inaccurate bin techniques and computations. This scheduling technique consists of two main objectives. First, to ensure that all applications are not exceeding the deadlines and that results in high quality output. Second, to decrease the time spent by execution of every workflow that causes cost to the user. The EDF-F-IC scheduling heuristic consists of two phases: tasks selection, and VM selection. In the first phase, priority is assigned to each task according to EDF policy. The task having the highest EDF priority is the task which has earliest

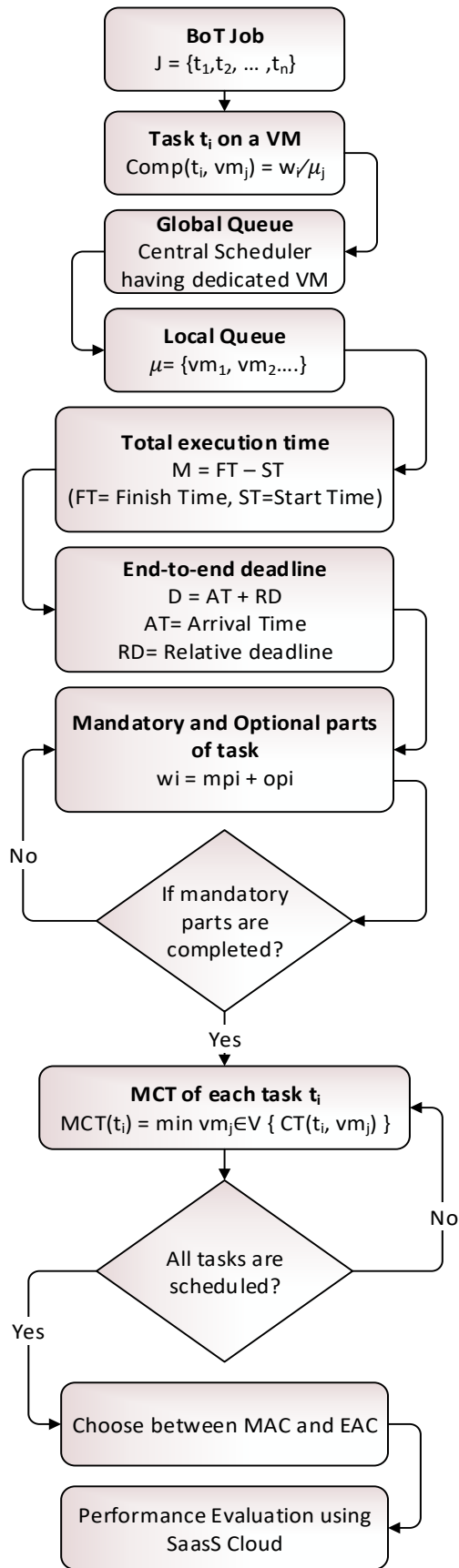


Figure 2.27 Workflow of BoT with approximate computation scheduling approach

end to end deadline. All the tasks are organized in ascending order of priorities. If there are two tasks and both have same deadlines, then both tasks are organized in descending order. In the second phase, when task selection is completed, then task is assigned to that VM which delivers it with the earliest estimated start time. The authors first find the initial position on which the ready task is located in the VM queue, agreeing to their priority so that least priority task does not precede the least one. Then they evaluate if the time saved by avoiding some part of the ready task is equal or greater than the total average time forced on the ready task sub-tasks. The proposed EDF-BF-IC technique is compared to EDF by considering communication workflow, intensive, and moderate application parameters.

#### **2.8.28 Hybrid Genetic and Cuckoo Search (HGCS) Algorithm**

The authors (Min-Allah. N et al., 2019) proposed a hybrid approach for scheduling real-time applications on cloud computing resources. The main performance parameters are total cost and makespan minimization. The schedulability of application is checked on cloud VMs. If application is feasible for execution, then the cost of execution is checked. The proposed HGCS algorithm selects low cost cloud computing resources where real-time applications are executed with total minimum execution time. The performance of the HGCS is compared with the other two well-known algorithms, the genetic, and cuckoo search.

### **2.9 RESOURCE ALLOCATION SCHEMES FOR REAL-TIME SERVICES IN EDGE COMPUTING**

When various things (devices etc.) and information are connected to the internet, it refers to the internet of things (IoT). Now a day's billions of IoT devices are



connected to the internet and huge amount of data produced by these devices need to be processed in a very short span of time. One solution to the huge data storage and processing is the cloud environment. As the distance between the IoT device and cloud increases, the transmission latency also increases, which also increase the response time. Furthermore, smart handheld devices have limited storage and computation capabilities which cannot accommodate applications demanding high storage and real-time processing (Zhang et al., 2020). The solution to this problem is the edge computing platform. The mobile edge computing technology helps in overcoming the limited memory and storage constraints problems by providing cloud computing facility adjacent to the smart IoT devices. The edge computing platform allows some application processing to be performed by small edge servers located between the IoT devices and the cloud in location curiously physically closer to the IoT device.

In edge computing, server is located in the edge network. The distance between the IoT devices and the computation resource is a single hop, due to which the latency is low, and the jitter is also very low. Edge computing is geo-distributed with location awareness and mobility support. Edge computing provides limited service scope with limited hardware capabilities to the mobile users. The storage capacity and computational power are also limited in edge computing. Figure 2.28 shows the general architecture of edge computing while Figure 2.29 portrays the taxonomy of edge computing presented by (E. Ahmed et al., 2017).

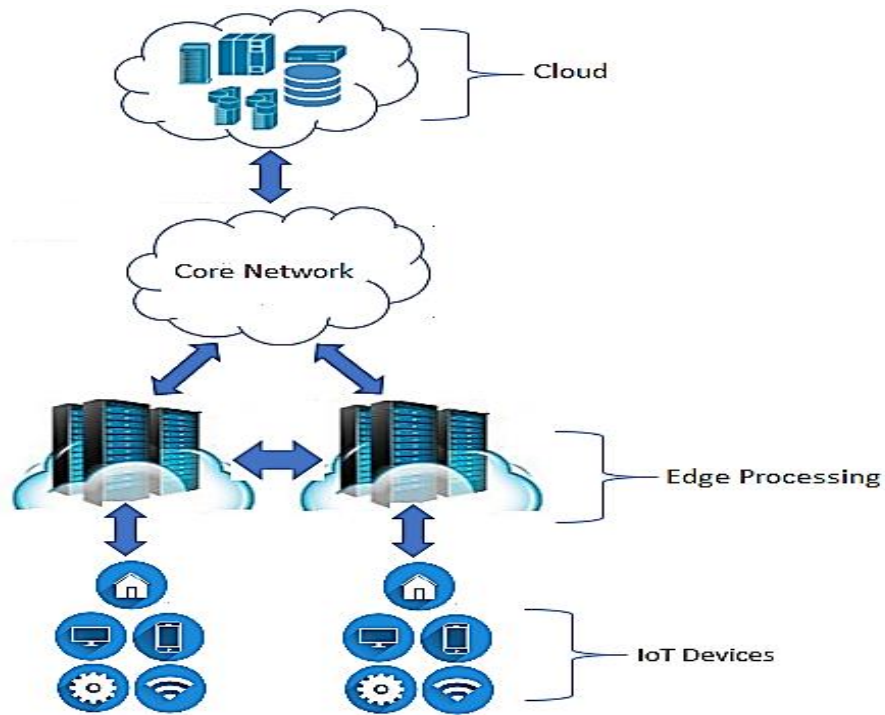


Figure 2.28 General architecture of edge computing

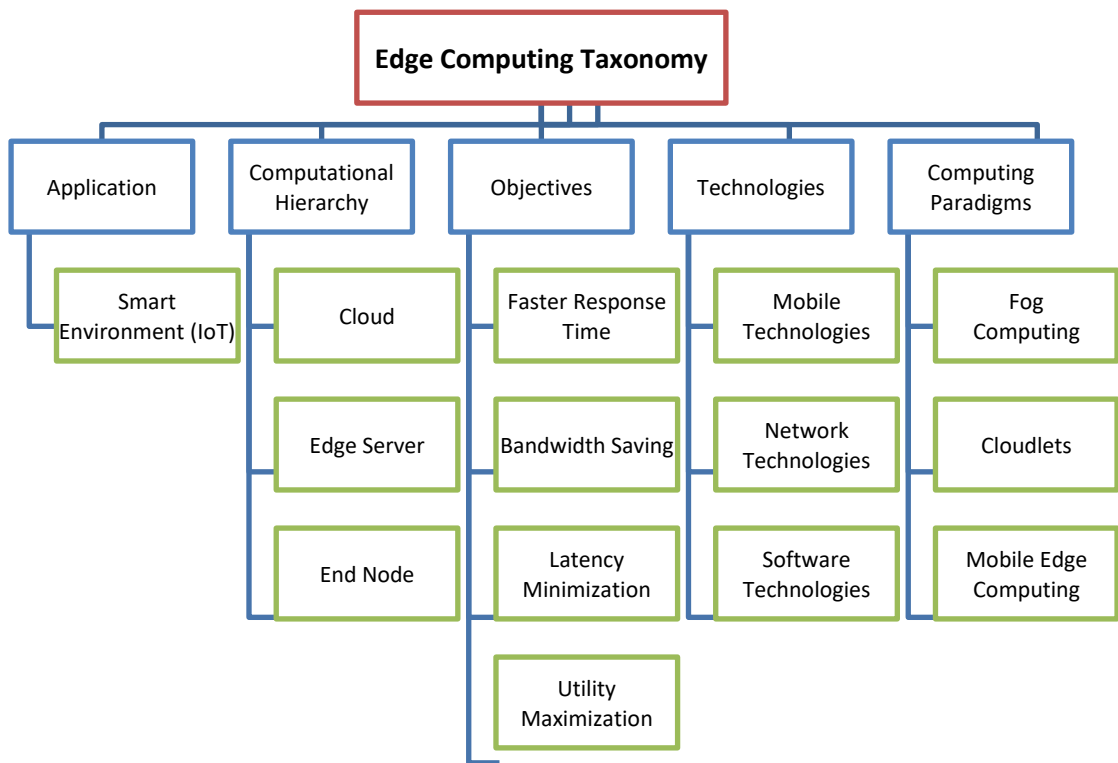


Figure 2.29 Taxonomy of edge computing

### **2.9.1 Resource Matching-based Allocation Scheme**

Due to the low latency in resource allocation, edge computing is becoming essential nowadays. There is a need to provide optimal solution in a cloud computing environment. (Hengliang. Tang et al., 2019) proposed an algorithm for dynamic resource allocation in edge computing environment. The resource allocation comprises of resource matching and scheduling processes on edge servers. To reduce network traffic and improve the user experience is a main part of every application. All information related to data is stored in memory for further use. In resource scheduling algorithm, the resources are scheduled with the help of edge orchestrator (EO). The traffic data is scheduled into the disk of edge servers (ESs). All the jobs are submitted to the corresponding edge server by the access point (AP) where every job contains multiple tasks. Now, the corresponding container for the tasks is configured by the edge server. Every edge server can present multiple containers containing RAM and CPU. Furthermore, each container executes single task at a time. Lastly, every edge server matches its tasks and containers by a Resource Matching (RM) algorithm. The proposed model presented better results than the existing techniques.

### **2.10 RESOURCE ALLOCATION SCHEMES FOR REAL-TIME SERVICES IN FOG COMPUTING**

With the increasing volume of IoT devices, the main problem faced by cloud computing is the latency. IoT devices need huge bandwidth for transferring huge amounts of data for communication as these devices send all the data to the cloud for processing and storage which consume bandwidth and energy. The processing of real-time data is not possible for the cloud due to its high latency.

The cloud model has insufficient ability to handle the IoT requirements in case of bandwidth, latency, and huge data volume produced by the IoT devices. Hence, there is a need to bring the cloud facility closer to the IoT devices to minimize the use of bandwidth and energy consumption and to reduce the latency.

CISCO introduced Fog computing to overcome the deficiencies of the cloud computing. Conceptually, Fog is the intermediate layer between the IoT devices and the cloud. The general architecture of the fog layer is represented in Figure 2.30.

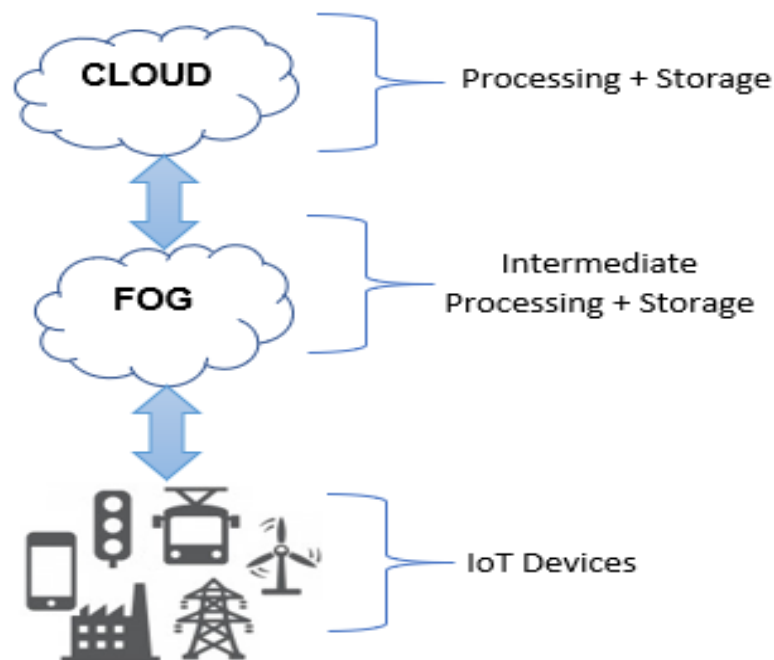


Figure 2.30 General architecture of fog computing

Fog is not the replacement of cloud due to the limited number of resources but its extension which is dominant in terms of lower service delay, processing cost, and response time. Similarly, augmented reality, virtual reality, and time sensitive applications which have rigid service delivery deadlines are also not efficiently execute on remote cloud resources (Sun et al., 2020). Using Fog computing, the

produced data is pre-analyzed on the asset, minimize the volume of data, and efficient management of run time behavior is carried out. Conversely, due to the wireless connectivity, power failure and devolved management, the failure ratio is high in the Fog computing (R. Mahmud et al., 2018). Naha et al., (2018) presented taxonomy of fog computing based on the requirements of application, infrastructure and platform which is presented in Figure 2.31.

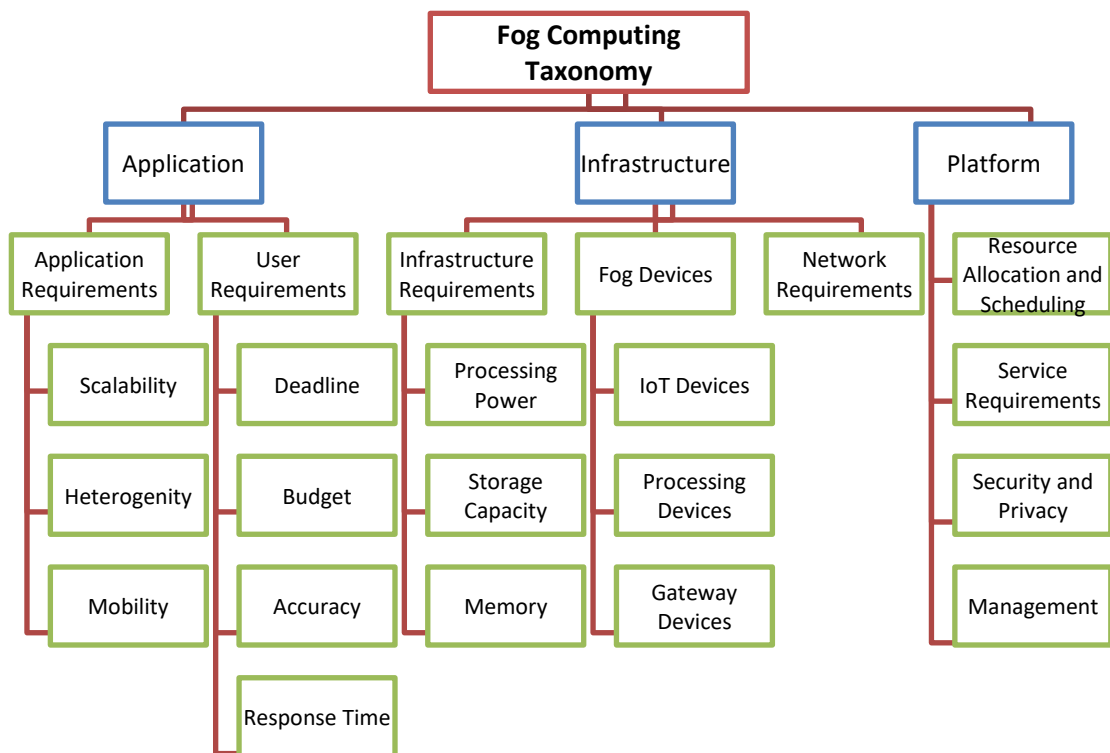


Figure 2.31 Taxonomy of fog computing

### 2.10.1 Hybrid Earliest Deadline First Approach

(Georgios et al., 2017) proposed hybrid technique to scheduled real-time workflows in a cloud computing environment. In fog computing, IoT workflows for dynamic scheduling in three-tiered architecture. The IoT comprises of devices and sensors

which transmit data via Wi-Fi network to the fog layer. The data arrive dynamically at central scheduler towards Poisson stream. Each job is non preemptible and denoted by a directed acyclic graph (DAG). The tasks are initially prioritized using EDF policy. Then, the tasks are scheduled and assigned to the VM according to earliest finish time (EFT) technique. The proposed model produced improved results as compared to fog-EDF. This general process is shown in Figure 2.32.

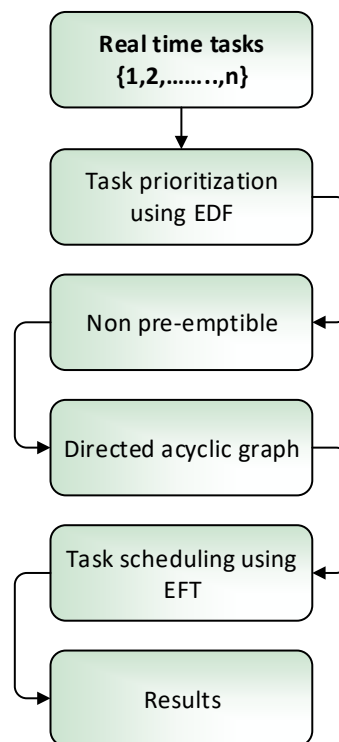


Figure 2.32 General process of hybrid EDF approach

### 2.10.2 Tasks Buffering and Offloading Policy

A task shows poor performance on the execution nodes due to limited capacity. Cloud computing resolves this issue by utilizing powerful resources which produce better quality results in reduced time. But it increases chances of data loss because several tasks transfer information over the network. To overcome this issue, (Lei. Li et al.,

2019) proposed an algorithm for processing heterogeneous real-time tasks. It comprises of three tiers to increase the QoS. Initially, the end tier is accessible by the data source of fog system. Secondly, the fog tier shows a minor latency during transmission between the fog and end devices. Thirdly, the cloud tier contains unlimited resource capacity to attain extra-ordinary performance. A single parallel virtual queue at the fog node is made to reduce time complexity. The proposed model performance is compared with the two resource allocation techniques, i.e., the round robin and maximum resource utilization. The results showed that greater value is achieved by a task when task buffering, and offloading policy is adopted. This scenario is depicted in Figure 2.33.

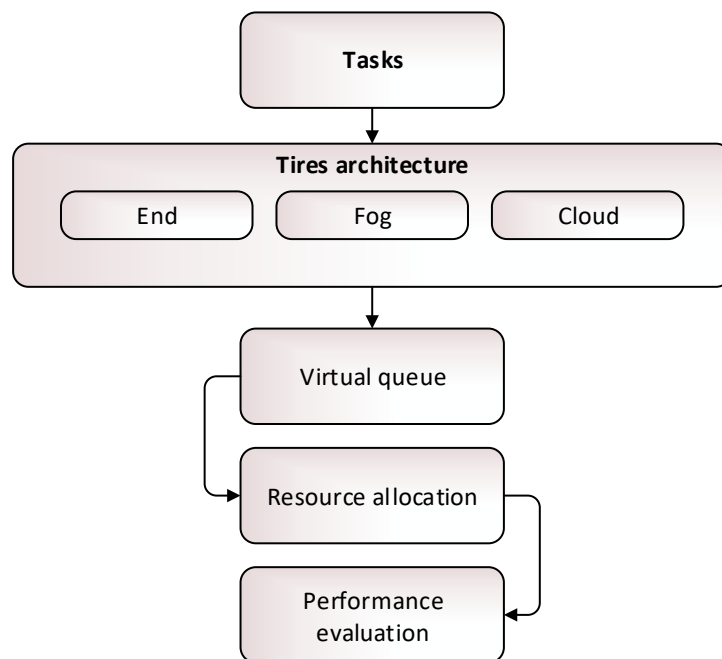


Figure 2.33 Task buffering and offloading scenario

## **2.11 RESOURCE ALLOCATION SCHEMES FOR REAL-TIME SERVICES IN MULTICORE SYSTEMS**

The RA schemes for real-time services in multicore environment are mainly categorized into online, offline, and mixed approaches (Zarrin et al., 2017). The online RA approaches map real-time applications dynamically at run-time without any prior knowledge of the system (B. Yang et al., 2013; Anagnostopoulos et al., 2013; F. Dong et al., 2006; G. Sabin et al., 2007). In offline RA approaches, the applications have advanced knowledge of the whole system and the resources status (Y. Jiang et al., 2008; D.W. Kim et al., 2002; J. E. Boillat et al., 1990; C. Marcon et al., 2007). These approaches have drawback of lack of knowledge how to solve variant application workload problem at runtime dynamically. In mixed approaches (H. Hoffmann et al., 2011; H. Shojaei et al., 2009; L. Schor et al., 2012), application mapping and selection is pre-calculated at design time.

### **2.11.1 Rate Monotonic Scheduling with Reduced Priority Levels Approach**

In reduced priority levels technique (Qureshi M. B et al., 2015), the hard-real-time tasks are grouped into multiple classes. Priority is assigned to a class instead of individual task which means that if there are total  $n$  tasks in a set, then the number of priority levels will be less than  $n$  which shows that one of the class has more than one task. The tasks within a class are prioritized based on the Rate Monotonic scheduling approach. This procedure reduces the assignment of priority levels.

### **2.11.2 Hybrid Cuckoo Search-based Algorithm**

(Xiangtao. Li et al., 2013) proposed hybrid cuckoo search (HCS) based algorithm to resolve permutation flow shop scheduling problem (PFSSP). The core purpose of the PFSSP scheduling is to minimize the makespan. Based on a random key, largest rank



value (LRV) rule is proposed to create a appropriate cuckoo search. Additionally, with the combination of Nawaz Enscore Ham (NEH) heuristic algorithm, a population with an assured value and range is generated. A Fast-Local Search (FLS) is injected to increase the local utilization for consideration. Finally, comparison is carried out for better performance of the HCS model with the PSOMA, OSA, PSOVNS, and HDE algorithms.

### **2.11.3 Online Accrued Scheduling Scheme**

(Shuo. Liu et al., 2011) proposed scheduling algorithm for real-time system on the task model online with the utility functions. Two different utility functions that are profit and penalty are associated with the task. If the task completes its execution within a specific time period, it is considered as profit. On the other hand, if the task misses its deadline, then it must pay a penalty. To schedule several real-time services requests, two non-pre-emptive heuristic scheduling are used. The purpose of using heuristics is to accept, schedule, or abort real-time services to maximize the sing time utility function. Initially, all the incoming real-time tasks are accepted to assess the requirement for the system. Then, all the incoming tasks in a ready queue are scheduled for execution. Lastly, reject awaiting requests and terminate the task when accomplished. The proposed model shows better results than EDF, GUS, risk-reward, and PP-aware scheduling algorithms. The high level working of the proposed scenario is depicted in Figure 2.34.

### **2.11.4 Least Feasible Speed (LFS) Technique**

(Nasro Min-Allah et al., 2012) introduced LFS technique for checking the schedulability of the real-time systems with minimum energy consumption procedure

on multicore systems. The authors suggested that the first point in a scheduling points set cannot guarantee the schedulability of the tasks with minimum power consumption. So, each task feasibility is checked by multiple scheduling points in a set. The proposed LFS technique outperforms the existing First Feasible Speed (FFS) counterpart. Furthermore, the authors have also suggested a strategy for load balancing on cores.

#### **2.11.5 Load Balancing by Tasks Splitting and Tasks Shifting Strategy**

Load balancing plays a vital role in improving multicore systems efficiency. In ref (Hussain. H et al., 2013), the authors distribute the real-time tasks on multicore systems in a way that the computational demand of tasks is fulfilled in specified deadlines and the total utilization of the core remains in bound. They perform utilization-based tests. When a core utilization becomes greater than 100%, then tasks are shifted to the other underutilized core.

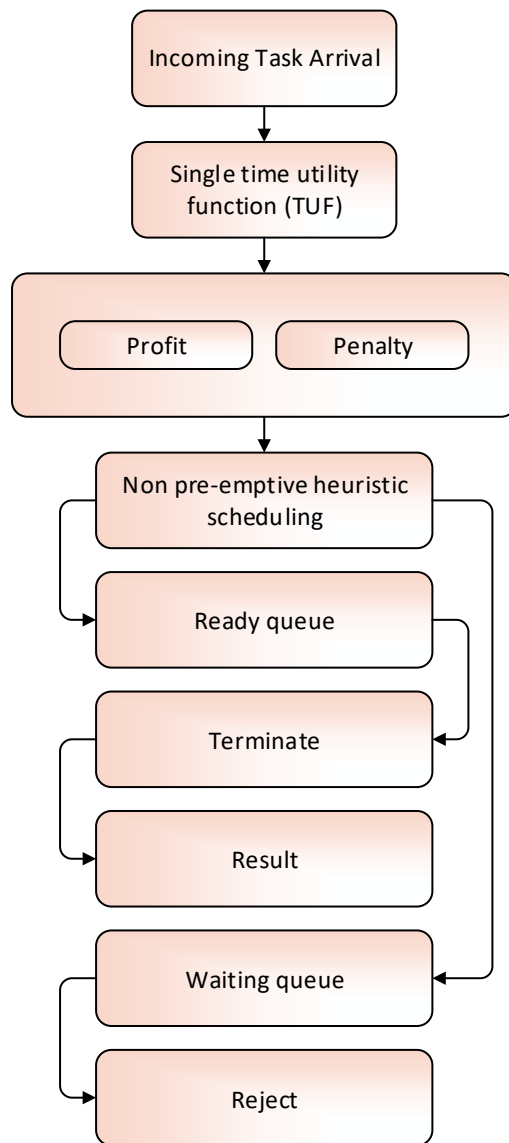


Figure 2.34 High level flow of online accrued scheduling scheme

### 2.11.6 Compatibility-aware Task Partitioning Scheme

It is essential to improve task execution within deadline constraints on multicore platforms. (Qiushi. Han et al., 2015) provide efficient tasks scheduled using rate monotonic scheduling (RMS) on multiple core platforms under fault tolerant requirements. In the proposed scheme, multicore scheduling is divided into two groups, namely, global and partitioning scheduling. In global scheduling, every task is executed on any core, while in partitioning scheduling, every task is assigned to a core

and all tasks are executed on that specific core. The authors considered partitioning scheduling because of low overhead. Only one task can be allocated at a time. Also, a task is assigned to a respective core which computes the compatibility. The task is re-executed when fault is identified. A check pointing scheme is merged with the compatible task to increase system utilization. Figure 2.35 shows this phenomenon.

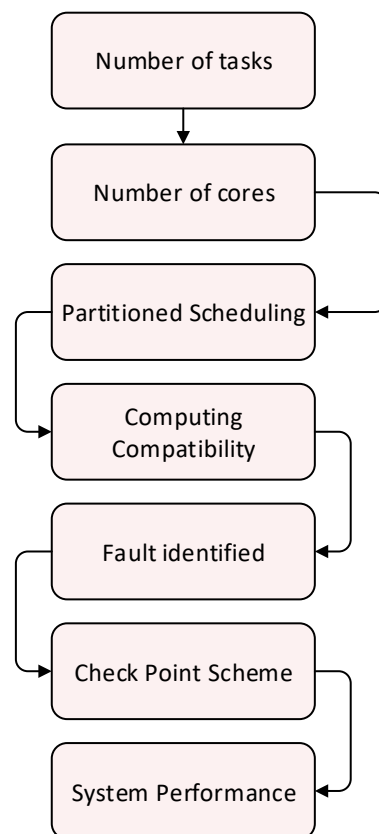


Figure 2.35 Compatibility-aware tasks partitioning scheme

### 2.11.7 Simple Combined Resource Usage Partitioning

The usage of reserve memory on the CPU is increased by some task running on the system due to which latency is also increased during the execution of tasks. To overcome this problem, (Gustavo et al., 2018) focused on the memory resource division for real-time systems. Initially, fixed priority is assigned to several tasks and

then the tasks are scheduled on multiple core platforms which share a single memory known as cache. The scheduling of hard real-time tasks is improved by co-scheduling memory and CPU. Then, memory bandwidth and memory latency are progressing using three dimensional systems with on-chip dynamic random access memory (DRAM). This is how tasks are scheduled to meet the deadline for the memories. The proposed model scheduled 19.5% more tasks than the other existing classic scheduling methodologies such as SCRUP and TATP.

### **2.11.8 Enhancing Shared Cache Performance-based Approach**

(P. Kumar et al., 2019) proposed task reprocessing scheme to enhance the performance of a common cache memory on a real-time multicore system (with quantum-placed universal recurring co-scheduling model) and to reduce the overhead in real-time scheduling by encouraging eligible task sets to reprocess based on heuristics called ENCAP (Enhancing shared Cache Performance). By utilization, processing rates are determined by splitting tasks into sub-tasks having pseudo cutoffs (intermediate cutoffs). These sub-tasks are processed using EDF scheme. Among sub-tasks, there exist cutoff miss or tie with the same cutoffs. The tie is broken by ENCAP.

The ENCAP is used to reschedule loads which are ignored due to the low priority load. Loads are assigned statically and processed based on a cache aware real-time EDF. Every load in a task is discharged regularly which is called an activity of the load. A periodic load can be derived by a slot which is partitioned between its consecutive load discharges and reprocessing cost. It implies the largest reprocessing time. Every activity of a load has a cutoff corresponds to the discharge time.

### **2.11.9 Large Time Demand Analysis Technique**

In scheduling points test, the basic feasibility of a task is determined by testing scheduling points for the lowest priority task first. But main drawback of such tests is that it is not possible to address the total execution demand of such tasks at a small number of points. Instead, the feasibility is determined at very large scheduling points set. Nasro Min-Allah (2019) performs the feasibility analysis of periodic real-time tasks by testing large number of scheduling points. At each scheduling point, the cumulative demand of a task is tested. The task is declared un-schedulable if at any point, the task execution demand is greater than the total processor capacity.

### **2.12 CHAPTER SUMMARY**

The resource allocation scheme in HPC systems plays a vital role in allocating resources to the user applications, especially when such applications have associated time parameters and need resources within specified time and user budget constraints. In this chapter, we have surveyed resource allocation schemes for real-time services in HPC paradigms, including grid, cloud, fog, edge computing, and multicore systems. Working of these schemes has been evaluated theoretically and presented pictorially so that a new researcher can be facilitated on a single platform. The detailed comparison based on common parameters provide an opportunity to easily find architectural and implementation related similarities and differences among different RA schemes. We have provided comprehensive analysis, which distinguishes this survey from the existing surveys in the RA domain. This survey specifically consolidated RA schemes only for real-time services which involve execution deadlines. In this chapter, RA schemes are classified based on real-time workloads and an overview of the most commonly used schemes was presented in large-scale

HPC systems. Each scheme was uncovered based on some common performance parameters.

## CHAPTER THREE

### PREDICTION-BASED RESOURCE ALLOCATION MODEL FOR REAL-TIME TASKS

#### 3.1 INTRODUCTION

In this chapter, we consider real-time tasks scheduling as an HPC resource management technique. The proposed resource allocation model considers real-time tasks which need data files for processing. Task requirements are specified and passed to the prediction analyzer module. The prediction analyzer predicts whether task execution will be completed within deadlines by considering all the task requirements. If the task is schedulable, then the scheduler schedule it on the feasible resources specified during prediction phase. In this way the resources are allocated to the feasible tasks.

In recent years, real-time systems scheduling on HPC platforms contributed huge volume to the plethora of scheduling theory. It has become the passable platform for executing computational-intensive applications on powerful resources (Abdullahi & Ngadi, 2016). The effectiveness of the real-time scheduling algorithm can be measured from the deadline miss ratio (Xie & Qin, 2005; Qureshi M. B et al., 2014) which can be calculated as follows.

$$\text{Deadline miss ratio} = \frac{\text{No. of tasks missing deadlines}}{\text{Total number of submitted tasks}}$$

Real-time scheduling algorithms are classified in to static and dynamic algorithms. In static scheduling algorithms, the tasks are characterized by static priorities which are assigned to the tasks before starting execution on computing



resources and remain unchanged throughout the execution of the tasks (Qureshi M. B et al., 2015). In dynamic scheduling, tasks priorities may change during execution.

### **3.2 PROPOSED RESOURCE ALLOCATION MODEL**

It can be observed that most of the existing solutions to the real-time data-intensive tasks allocation problem in HPC domain leave gaps for deadline miss. The data and communicational aspects between computing and storage resources and specification of the feasible resources and task execution time on these resources is unaccounted in the existing resource optimization settings. The advanced feasibility testing of the real-time tasks under fixed priority scheduling technique has always been challenging when data constraints are considered. From data files processing perspective, it has also been of interest to include data files transfer time in feasibility analysis on computational resources when some of the files are locally available to the tasks and some to be transferred from the remotely located storage resources.

In this chapter, a novel and fine-tuned resource allocation model for real-time application on HPC resources is developed to cope with the aforementioned problems. The proposed model is referenced in Figure 3.1.

In the proposed model, batch of real-time tasks is submitted to the Tasks Demand Specifier module which specifies the basic tasks parameters. The parameters include required execution time, period, deadline and number of required data files. After specifying the parameters, the batch is forwarded to the Offline Prediction Analyzer which analyzes tasks in advance for feasible allocation. In this module, computational resources ranking strategy is applied which guarantees tasks execution within deadline with minimum possible time. The ranking technique helps in selecting the most appropriate resources for application scheduling. The tasks grouping

approach based on some common parameters is adapted to achieve schedulability without compromising tasks deadlines constraints. By positive negative points technique, the points on which task schedulability is feasible are identified and rest are discarded. The points are then passed to the Prediction Analyzer where task's feasibility is checked on each positive point. When tasks are declared suitable, then they are assigned execution priorities by using priority assignment technique. When tasks are prioritized then the batch is forwarded to the scheduler which then select resources from a pool of resources and allocate to the tasks. If tasks are declared not schedulable on resources by Offline Prediction Analyzer, then the batch of tasks is discarded, and further evaluation is not performed. The advantage of the prediction analysis stage is to save time by refraining further analysis on non-schedulable tasks.

### **3.3 MATHEMATICAL MODELING AND PROBLEM FORMULATION**

#### **3.3.1 Proposed Task and Resource Model**

The proposed model is characterized by the following modules. We consider a real-time application which is composed of multiple tasks. The application considered here is termed as metatask  $\Gamma$ . Each task  $i$  has pre-defined parameters; execution time  $c_i$ , needed by a task  $i$  to execute on a computational resource, period  $p_i$ , which is the time interval between the activation of two consecutive jobs of a task  $i$ , and deadline  $d_i$  which is the time by which the task  $i$  should complete its execution. The task model can be expressed mathematically as,

### 3.3.2 Basic Task Model

$$\tau_i(c_i, p_i, d_i) \quad (3.1)$$

where

$p_i$ : period of task  $i$

$c_i$ : computation time of task  $i$

$d_i$ : deadline of task  $i$

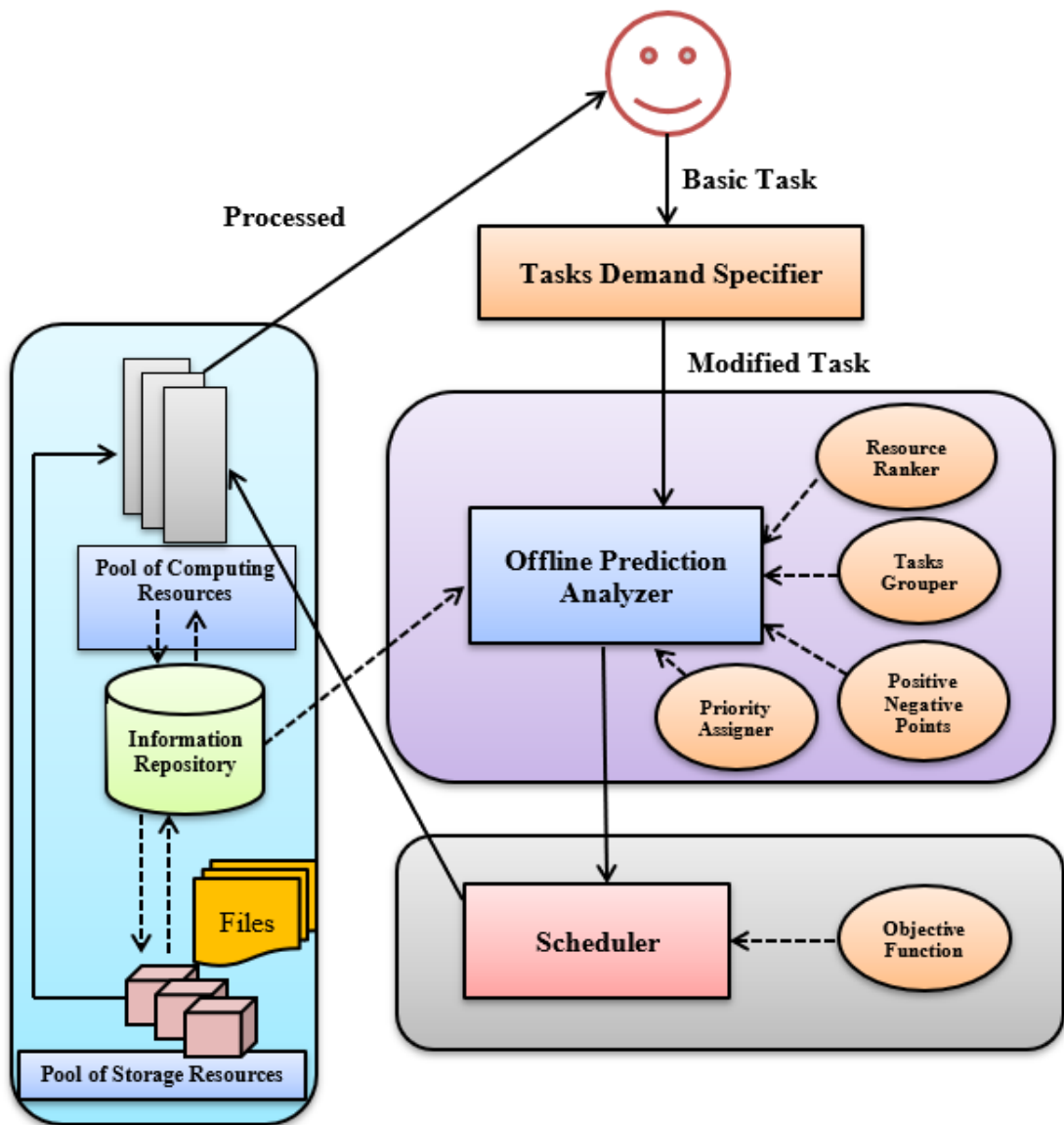


Figure 3.1 Proposed prediction-based resource allocation model

In our proposed model, we are considering tasks with data requirements. We assume that the data files are replicated on multiple data storage resources. By considering the data requirements, the modified task model is represented by Equation. 3.2.

### 3.3.3 Modified Task Model

$$\tau_i(c_i, p_i, d_i, F_i) \quad (3.2)$$

where

$F_i$ : Set of files required by task  $i$

$F_i \subset F \{f_1, f_2, \dots, f_k\}$  where  $F$  is set of total  $k$  files located on distributed data storage resources.

The metatask is a set of total  $n$  tasks represented by Equation 3.3.

### 3.3.4 Metatask $\Gamma$

$$\Gamma = \{\tau_1, \tau_2, \tau_3, \dots, \tau_n\} \quad (3.3)$$

The feasibility of the metatask is determined by individual tasks. If all the tasks (1, 2, ...,  $n$ ) in a metatask are feasible, then a metatask is schedulable.

### 3.3.5 Offline Prediction Analyzer

This function analysis the feasibility of a task and determine in advance whether the task will fulfill deadline if allocated to a particular resource by considering task and user constraints. The Offline Prediction Analyzer component take into consideration the following sub-components for predicting task's feasibility on resources.

### 3.3.5.1 Positive Negative Points Set for Task $i$

$$PNP_i = \{x.p_j | j = 1, \dots, i; x = 1, \dots, \lfloor \frac{p_i}{p_j} \rfloor\} \quad (3.4)$$

where  $p_j$  is the period of the higher priority task than task  $i$ .

From the  $PNP$  set, the positive points (time instants) are identified by checking task's feasibility, and the rest of the points are declared as negative points on which the task is infeasible. This technique will help in minimizing the makespan because the negative points will be further excluded from the task feasibility analysis.

### 3.3.5.2 Resource Demand Calculator

$$RD_i(T) = c_i + RD_z(T); T \in PNP_i \quad (3.5)$$

$$RD_z(T) = \sum_{j=1}^{i-1} \lfloor \frac{T}{p_j} \rfloor c_j \quad (3.6)$$

where

$RD_i(T)$ : Resource demand of task  $i$  at time  $T$

$RD_z(T)$ : Resource demand of all higher priority tasks than task  $i$  at time  $T$ .

The resources are HPC heterogeneous resources each one is equipped with pre-defined processing power. The resources are distinguished based on their architecture.

### 3.3.5.3 Execution Time of Task $i$ on Computing Resource $r$

$$ET_{ir} = \frac{RD_i(T)}{pow_r}; r \in R \quad (3.7)$$

where

$R$ : set of computing resources

$pow_r$ : power of resource  $r$  in MIPS

### 3.3.5.4 Resource Ranking Function

$$\text{Rank} = pow_r + B_{rd} \quad (3.8)$$

where

$B_{rd}$ : bandwidth of computing resource  $r$  to data storage resource  $d$

The rank of the computing resource is calculated for each task. If a single file is replicated on more than one storage resources, then the rank of a single computing resource  $r$  where the task is initially feasible will be calculated to all the storage resources on the required file is stored. The file will be retrieved from the storage resource for which the rank value is high. Figure 3.2 illustrates this scenario.

For example, file  $f_1$  is stored on both data resources  $d_1$  and  $d_2$  which is required by the task executed on computing resource  $r$ . The file will be fetched from the data resource for which the corresponding rank is high. Also, if a task is feasible on more than one computing resources, then the computing resource for which the rank is high will be selected for task execution while ties are broken arbitrary.

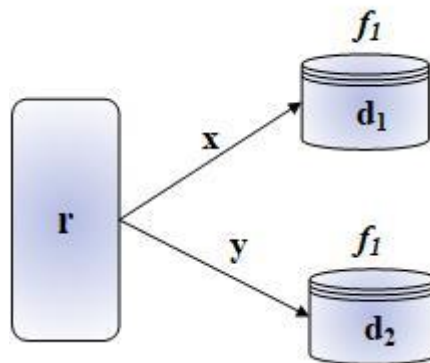


Figure 3.2 Resource rank calculation

### 3.3.5.5 Tasks Grouping

$$G_k = \{\tau_i, \tau_{i+1}, \dots, \tau_n\} \quad (3.9)$$

where

$G_k$ : Group  $k$  of  $n$  tasks

Tasks are grouped based on common demands or based on priorities. For example, if tasks are grouped based on common files demands and there are 10 tasks in a set and total 5 files are available on storage resources. If  $\tau_1, \tau_4, \tau_5$  need files  $f_1$  and  $f_5$ ,  $\tau_2, \tau_3, \tau_6, \tau_7$  need files  $f_2, f_3, f_4$ ,  $\tau_8, \tau_9, \tau_{10}$  need files  $f_1$  and  $f_3$ , then tasks can be grouped into 3 groups.

$$G_1 = \{\tau_1, \tau_4, \tau_5\}$$

$$G_2 = \{\tau_2, \tau_3, \tau_6, \tau_7\}$$

$$G_3 = \{\tau_8, \tau_9, \tau_{10}\}$$

### 3.3.5.6 Priority Assigner

This function assigns priorities to the tasks using some traditional algorithm like Rate Monotonic (RM).

RM priority assignment:

$$\text{priority}(\tau) \propto 1/\text{period}(\tau) \quad (3.10)$$

### 3.3.5.7 Objective Function

The Quality of Service (QoS) parameters that may be considered, are

- a) *makespan minimization*
- b) *cost efficiency*
- c) *maximum resource utilization etc.*

Task constraints are completing task and metatask within deadlines. If all tasks within a metatask are completed within deadlines by considering all the task and user constraints, then the metatask is completed within deadline.

#### ***3.3.5.8 Resource Model***

In our proposed model, we consider computing, storage and network resources. By network resource we mean the bandwidth allocated for transferring data files from storage locations to the computing resources. The resources are heterogeneous in nature which means that they have different architectures and different processing powers.

By considering all the task's demands and concerned requirements for the resources, the prediction analyzer predicts whether the given task is schedulable on some computing resources. If the task is schedulable, then it is forwarded to the scheduler for further allocation.

### **3.4 CHAPTER SUMMARY**

In this chapter, we have proposed prediction-based model for allocating HPC resources to real-time tasks. We assumed that the tasks need data files for completing their processing on computing resources. The resources are computing, storage and network resources. The files are transferred from the far away located data resources to the computing resources which consume network bandwidth. The prediction analyzer considers different parameters and predicts whether, the arrived task will meet the deadline or not. The advantage of this model is to reduce the overall processing time of the real-time tasks. The model was mathematically formulated.



## **CHAPTER FOUR**

### **EFFICIENT RESOURCE ALLOCATION TECHNIQUE FOR REAL-TIME DATA-INTENSIVE TASKS IN CLOUD COMPUTING SYSTEMS**

#### **4.1 INTRODUCTION**

Cloud computing is the de facto platform for deploying resource-intensive tasks due to the collaboration of large-scale resources operating in cross-administrative domains. The cloud computing resource scheduling and allocation is a crucial issue in achieving efficient utilization of available resources especially when resource-intensive tasks have real-time deadlines and data requirements. Traditional approaches are sufficient only when data resources are located locally, since data is available at the computing resources for tasks execution. However, the computational cost and execution time of the resources has not been thoroughly deliberated when data storage resources are remotely located. In such approaches there is a chance that some tasks may miss their deadlines during execution due to the urgency of the tasks and limited budget constraints. The timing requirements exacerbates efficient task scheduling and resource allocation problem. To cover the aforementioned challenges, we propose a time and cost-efficient resource allocation strategy for real-time data-intensive tasks in the cloud computing environment which minimizes the data files transfer overhead to computing resources where the tasks are executed by selecting appropriate computing and storage resources. The proposed algorithm considers heterogeneity of resources and data files transfer time and cost in task's feasibility analysis when files are replicated on remotely located storage resources. The celebrated results show the effectiveness of the proposed technique in terms of resource selection and tasks

processing within time and budget constraints when compared with the RDTA (Martel, 2020) and Greedy algorithms.

With fast growing advancements in IT industry, the real-time applications are handy candidates for utilizing computing power in cloud computing environment in order to maintain deadline constraints. In addition, the cloud storage resources provide facilities such as accommodating data replication to satisfy data requirements of the data-intensive real-time applications that need to access, process and transfer data files stored in distributed data repositories (Venugopa & Buyya, 2008). Examples of such applications are self-driving vehicles which depend on the data and computations under a complex network of interconnected devices such as GPS, surveillance cameras, radar, laser light, odometry, etc. to perceive the surroundings (Martel, 2020; Statista, 2020). The cloud providers such as Amazon EC2 (Amazon, 2020) provide computing facilities (virtual machines) on pay-as-you-go basis at the rate of 10 cents per hour. The lease prices vary depending on the virtual machines (VMs) specifications. The normal VM offers an approximate processing power of 1.2 GHz Opteron processor with storage capacity of 160 GB disk space and 1.7 GB of memory (Amazon, 2020; Li et al., 2012; Armbrust et al., 2009). Such facilities pave the way for executing time critical and IoT applications which demand high processing and storage capabilities (Amadeo et al., 2017). The IoT devices offload many tasks to the cloud environment from the smart systems because these devices have very limited processing and storage capabilities. Leveraging the capabilities of virtualization technology, VMs can be scaled up and down depending on the current system workloads (Chen et al., 2015). However, there is a lack of efficient resource scheduling and allocation strategies for deploying real-time applications with stringent QoS and data requirements in cloud computing environments.

The Global Institute report on analyzing economic impact of IoT devices show that it will increase upto \$11 trillion by the year 2025 (Mckinsey, 2015). This increase is because the IoT devices and smart homes appliances ranging from small sensors to large scale biometrics offload data and computation to the cloud computing environment on regular basis. For this purpose, the IT companies provide solutions such as Apple's HomeKit (Davidson, 2017), Samsung's SmartThings (Ambient, 2020), Amazon's Alexa (Amazon, 2020), and Google's Home (Google, 2018), etc.

The driving force of the cloud computing is virtual machine manager (VMM) that creates the virtual resources of the physical machines. The basic functionality of the VMM is to separate virtual computing environment from the underlying physical infrastructure. In this research work, we implement the rate monotonic (RM) scheduling policy to allocate cloud computing resources to the real-time periodic tasks. The scheduling problem is divided into three parts: the processing environment (cloud virtual machines), the nature of real-time task (fixed priority system), and the optimization criteria (time and cost-efficient allocation). The real-time task set is a collection of multiple tasks, each of which requires data files for processing. The required data files are requested from the remotely located storage resources. The intelligent selection and assignment of cloud computing resources is investigated while data files are replicated on decoupled storage resources and accessed by utilizing networks of varying capabilities. These files are fetched to the computing resources where tasks are executed which add transfer time to the tasks total execution time.

The major research contributions of this chapter are as follows.

1. Creating a model for selecting appropriate cloud computing and storage resources to execute tasks with timing and data constraints when data files are replicated on multiple resources,
2. Partitioning the task sets into groups based on common data-files demands such that timing constraints of the original tasks set do not disturb,
3. Allocating cloud computing resources to periodic tasks such that the overall timing constraints remain intact.

## 4.2 TASK, RESOURCE AND COST MODELS

In this research, we consider scheduling feasibility of real-time periodic tasks in an HPC environment. Our concerned HPC environment is composed of both computational and storage resources located remotely and connected by network links of different bandwidths. The resources are heterogeneous and characterized by power and cost constraints. In this chapter, we concentrate on two basic constraints; (a) the real-time tasks deadlines, and (b) user specified budget. The presented model extends the RDTA model (Martel, 2020) by introducing cost parameters, data files replication scenario, and tasks grouping criteria.

### 4.2.1 Task and Resource Model

We consider batch processing of real-time periodic tasks each of which can generate infinite number of jobs. In periodic tasks set  $T = \{task_1, task_2, \dots, task_n\}$ , each  $task_k$  is defined by the quadruple:

$$task_k = (r_k, e_k, d_k, period_k) \quad (4.1)$$

where  $r_k$  shows the release time of the first job,  $e_k$  the required computation time,  $d_k$  the relative deadline of  $task_k$  which is the time difference between the

absolute deadline and release time of a job, and  $\text{period}_k$  the period which is the time different between the two successive jobs of a task  $k$ .

In the above discussed model, a job  $i$  released at time instant  $r_k + (i - 1) \cdot \text{period}_k$  needs to execute for  $e_k$  units before the time  $r_k + (i - 1) \cdot \text{period}_k + d_k$ . In our task model, we concentrate on constrained deadline model which assumes that  $d_k \leq \text{period}_k, \forall k \in T$ . Tasks preemption is not allowed and context switching overhead is subsumed into  $e_k$ . We also assume that  $r_k = 0, \forall k \in T$  which means that feasibility of tasks is checked when the system is most loaded.

We consider computing resource set  $CR$  such that  $CR = \{CR_1, CR_2, \dots, CR_r\}$  each one is characterized by a computing power  $CP_y$  ( $1 \leq y \leq r$ ) such that  $CP_y \in CP$  where  $CP = \{CP_1, CP_2, \dots, CP_r\}$  and measured in Millions of Instructions per Second (MIPS). The execution time of a  $task_k$  on resource  $CR_y$  can be computed by

$$EET_{ky} = \frac{e_k + e_{\text{higher}}}{CP_y} \quad (4.2)$$

where  $e_{\text{higher}}$  is the execution time of the higher priority tasks than  $task_k$ .

Mathematically,

$$e_{\text{higher}} = \sum_{j=1}^{k-1} \left\lceil \frac{t}{\text{period}_j} \right\rceil e_j, \quad t \in PNP_j \quad (4.3)$$

where  $PNP$  set accumulates points or time instants on which task feasibility is analyzed. The  $PNP$  set is defined as follows.

**Definition 1.**  $PNP_k$  is a set of positive and negative points for  $task_k$  constituted by the relation  $x \cdot \text{period}_j$  such that  $1 \leq j \leq k$  and  $1 \leq x \leq \lfloor \text{period}_k / \text{period}_j \rfloor$ , where  $\text{period}_j$  represents periods of higher priority tasks than  $task_k$ . The point  $t_p \in PNP_k$  is said to be positive if task  $k$  is declared feasible (i.e., completes its execution before

deadline) at some point  $t$  by considering all associated time and data constraints. The point  $t_N \in \text{PNP}_k$  declares the  $task_k$  infeasible when it misses the deadline. Each point in  $\text{PNP}_k$  is called rate-monotonic scheduling point.

From Definition (1), it is concluded that the set  $\text{PNP}$  is the union of positive points set  $\text{PP}$  and negative points set  $\text{NP}$  where  $\text{PP} = \text{PNP} - \text{NP}$  and  $\text{P} = \text{PNP} - \text{PP}$ . In other words,  $\text{PNP} = \text{PP} \cup \text{NP}$ .

#### 4.2.2 Data Files Model

In our task model, a tasks set  $T = \{task_1, task_2, \dots, task_n\}$  consists of data-intensive real-time tasks where each task  $task_k$  needs set of data files  $DF_k$  for its execution. The set  $DF_k = \{f_{k1}, f_{k2}, \dots, f_{km}\} \subseteq DF$ . The file  $f_{kx} \in DF_k$  is stored on data storage resource  $dr_w$  where  $dr_w \in DR_k$  and  $DR_k \subset DR$ . The  $DR$  is the set of total storage resources in the HPC environment. In other words, files in  $DF_k$  are stored on  $DR_k$  storage resources. We assume that the data files are replicated on more than one data storage resources.

The total execution time  $TT$  of a task  $task_k$  is the sum of actual computation time  $EET$  of  $task_k$  on computing resource  $CR_y$  and the transfer time taken by the required  $m$  data files in the set  $DF_k$  by transferring from storage resources  $DR_k$  to the computing resource  $CR_k$  where task  $k$  is executed. Mathematically,

$$TT_k = EET_{ky} + \sum_{z=1}^m FT_{f_{kz}} \quad (4.4)$$

where  $FT_{f_{kz}} = \mathbb{R}_w + \text{Size}_{f_{kz}} / \text{BW}_{wy}$  is the transfer time of the file  $f_{kz}$ .

The  $\mathbb{R}_w$  represents the response time of the data storage resource  $dr_w \in DR_k$  where the data file  $f_{kz}$  is stored. The response time is the time when the request to

fetch the file is made to the time when the request is entertained. Algebraically, response time of data resource  $dr_w$  is calculated as:

$$\mathbb{R}_w = ST_{f_{kz}} + WT_{f_{kz}} \quad (4.5)$$

where  $ST_{f_{kz}}$  is the service time and  $W_{f_{kz}}$  is the waiting time of the request respectively for accessing the file  $f_{kz}$ . Also, the  $Size_{f_{kz}}$  denotes the size of the file  $f_{kz}$ , and  $BW_{wy}$  shows the link bandwidth between data storage resource  $dr_w$  and computing resource  $CR_y$ . The proposed model selects that storage resource for file access for which  $FT$  is minimum i.e.,  $min(FT)$ .

### 4.2.3 Tasks Grouping

The data-intensive real-time tasks in  $T$  are grouped into  $x$  groups based on common data files demands. The tasks in a group represent subset of  $T$  or we can say each group is a set of tasks for easy understanding. The task grouping taxonomy is pictorially represented in Figure 4.1.

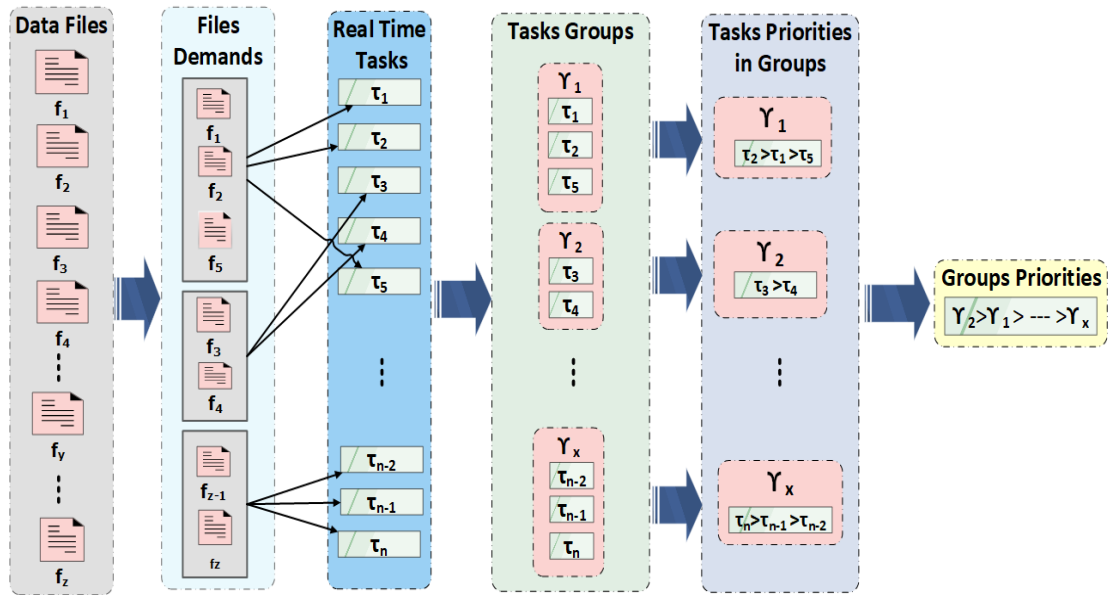


Figure 4.1 Tasks grouping taxonomy

Based on real-time tasks grouping criteria, the group of tasks, its cardinality, and priority assignment is defined in the following sections.

**Definition 2.** A group of real-time tasks  $\Upsilon_x$  is a subset of tasks i.e.,  $\Upsilon_x \subseteq T(x \leq n)$  having common data files demands. Each group  $\Upsilon_x$  contains minimum one task which concludes that  $\Upsilon_x \neq \{\}$ .

**Definition 3.** The cardinality of a task group defines total number of tasks in a group. Let there are total  $x$  number of groups then cardinality of the original tasks set  $T = \{\text{task}_1, \text{task}_2, \dots, \text{task}_n\}$  can be defined as:

$$\text{card}(T) = \sum_{i=1}^x \text{card}(\Upsilon_i) \quad (4.6)$$

The advantage of the tasks grouping mechanism is to reduce the total number of priority levels (Qureshi et al., 2015). Additionally,  $t_N \in NP$  for a higher priority task  $\text{task}_j \in \Upsilon_x$  remains the member of  $NP$  for all lower priority tasks than  $\text{task}_j$  in  $\Upsilon_x$  since tasks in the same group are also sorted on the basis of RM priorities. In this way a smaller number of points is tested in the  $PNP$  set which decreases the execution time.

From the above definitions, the following can be observed,

**Observation 1.** Let each tasks group  $\Upsilon_i$  constituted from task set  $T = \{\text{task}_1, \text{task}_2, \dots, \text{task}_n\}$  contains a single task in worst case scenario and  $x$  represents total number of groups in the system, then  $x = n$

where  $n$  represents  $\text{card}(T)$ .

Each real-time task in our task model has deadline  $d$  which demonstrates the maximum allowed time for a task to complete its execution on a single computing



resource. Let the maximum and minimum deadlines of the tasks in a group  $Y_x$  are denoted by  $d_{\max}$  and  $d_{\min}$  respectively. Here an interesting observation can be made.

**Observation 2.**  $d_{\max}$  of the tasks  $\{\text{task}_1, \dots, \text{task}_l\} \in Y_x (l \leq n)$  sorted by RM priority assignment technique and following the implicit deadline model (where period = deadline) is the period of the last task while  $d_{\min}$  is the period of the first task in  $Y_x$ . It follows the relation:

$$\begin{aligned} d_{\max} &= \text{period}_l \\ d_{\min} &= \text{period}_1 \end{aligned} \quad (4.7)$$

where  $\text{period}_l$  and  $\text{period}_1$  represent periods of the last and the first task in  $Y_x$  respectively.

**Proof:** The RM technique sorts the tasks based on priorities assignment criterion: *the lesser is the period of the task, the higher is the priority*. It means that the last task  $\text{task}_l \in Y_x$  has the lowest priority and first task  $\text{task}_1 \in Y_x$  has the highest priority among all tasks in  $Y_x$ . The  $\text{task}_1$  is executed in the first and  $\text{task}_l$  is executed in the last slots. In other words,  $\text{priority}(\text{task}_1) \geq \text{priority}(\text{task}_2), \dots, \geq \text{priority}(\text{task}_l)$  which follows that  $\text{period}(\text{task}_1) \leq \text{period}(\text{task}_2), \dots, \leq \text{period}(\text{task}_l)$ . It also follows that  $d_{\max} = \text{period}_l$  and  $d_{\min} = \text{period}_1$  which completes the proof.

Liu et al., (1973) discussed that rate-monotonic (RM) assigns static priorities to tasks and considered as optimal scheduling algorithm among static priorities assignment scheduling algorithms. By optimal they mean that RM should must schedule a task if any other static priority assignment algorithm can schedule that task. Following are the general characteristics of RM scheduling technique which play role in proving its optimality.

1. the system should consist of fixed number of tasks;
2. the tasks should have execution times known in advance;
3. each task has completion deadline equal to its period;
4. tasks should be periodic which means that instances or jobs of a task should arrive after a fixed time interval;
5. tasks should be independent;
6. all tasks should arrive at time 0. This time instant is also known as critical instant and the system is considered as the most overloaded at this instant.

**Definition 4.** The period of a tasks group is defined as the temporary period attached to the group of tasks which is the period of the last task in a group. In other words,  $period_1$  is the group period because the tasks in a group are sorted using RM technique. For example, if a group  $Y_x$  accommodates tasks  $\{task_1, task_2 \dots, task_1\}$  , then

$$period(Y_x) = period_1 \quad (4.8)$$

Since the groups are sorted on the basis of RM priorities, so  $period(Y_1) \leq period(Y_2), \dots, \leq period(Y_1)$  which states that  $priority(Y_1) \geq priority(Y_2), \dots, \geq priority(Y_1)$ . Equation (4.8) states that all tasks in  $Y_x$  must complete execution at or before  $period_1$  . It is further evaluated that since the tasks in the same group  $Y_x$  are also sorted by RM priorities, so  $t_N \in NP$  for a higher priority task  $task_i \in Y_x$  remains the member of  $NP$  for all the lower priority tasks than  $task_i$  in  $Y_x$  .

**Definition 5.** The group capacity can be defined as the total number of tasks in a group. Tasks in a group are added based on common data files demand. The group capacity can be analyzed based on resource utilization by a group of tasks called

group utilization (GU) which is defined as the sum of the resource utilization of each task in the group. The computing resource utilization of each task is termed as the task utilization (TU). Let  $n$  denotes the total number of tasks in a group  $Y_x$ , then GU and TU can be found as follows.

$$\begin{aligned} GU_{Y_x} &= TU_1 + TU_2 + \dots + TU_n \\ &= \sum_{i=1}^n TU_i \end{aligned} \quad (4.9)$$

where

$$TU_i = \frac{e_i}{\text{period}_i} \quad (4.10)$$

**Theorem 1.** Let  $Y_x$  is a group of  $n$  periodic tasks, where each task is characterized by TU. The  $Y_x$  is RM feasible if the following condition holds.

$$GU_{Y_x} \leq n(2^{1/n} - 1) \quad (4.11)$$

The inequality (11) is called as Liu & Layland (LL) test reported by (Liu et al., 1973). The expression  $n(2^{1/n} - 1)$  is the threshold value of a group which means that a group  $Y_x$  can accommodate tasks as for as the  $GU$  remains lower than or equal to the threshold value. Equation (4.11) is checked repeatedly when a new task is added to the group. If adding a task changes the inequality  $GU > n(2^{1/n} - 1)$ , then the incoming tasks are added to another group. The authors in [2] refer the LL test as the sufficient condition test. They claim that it is not necessary that the tasks in a group are not feasible for scheduling if Equation (4.11) does not hold. It means that utilization-based tests are enough only but not necessary. Let us explain by the following example task set taken from (Min Allah Nasro & Khan Samee, 2011; Liu, 2000).

**Example 1.** Consider a tasks group  $Y = \{task_1, task_2, task_3, task_4\}$  where tasks follow RM ordering and having following characteristics,

<b>Tasks</b>	$c_i$	$period_i$
$task_1$	2	3
$task_2$	1.5	6
$task_3$	0.5	12
$task_4$	1	24

The TU and GU values for Y are;

$$TU_1 = 0.666$$

$$TU_2 = 0.250$$

$$TU_3 = 0.041$$

$$TU_4 = 0.041$$

$$GU_Y \cong 1$$

$$\text{threshold} = 0.756$$

It shows that the example 1 tasks group is not schedulable by using LL test because it does not satisfy Equation (4.11). But the Gantt Chart in Figure 4.2 shows the schedulability of these tasks within deadlines under RM technique.

From the above discussion it is clear, that LL test is sufficient only, so we use LL test for checking group capacity only. For analyzing task or group feasibility, we use positive-negative points (*PNP*) test which is necessary and sufficient conditions test.

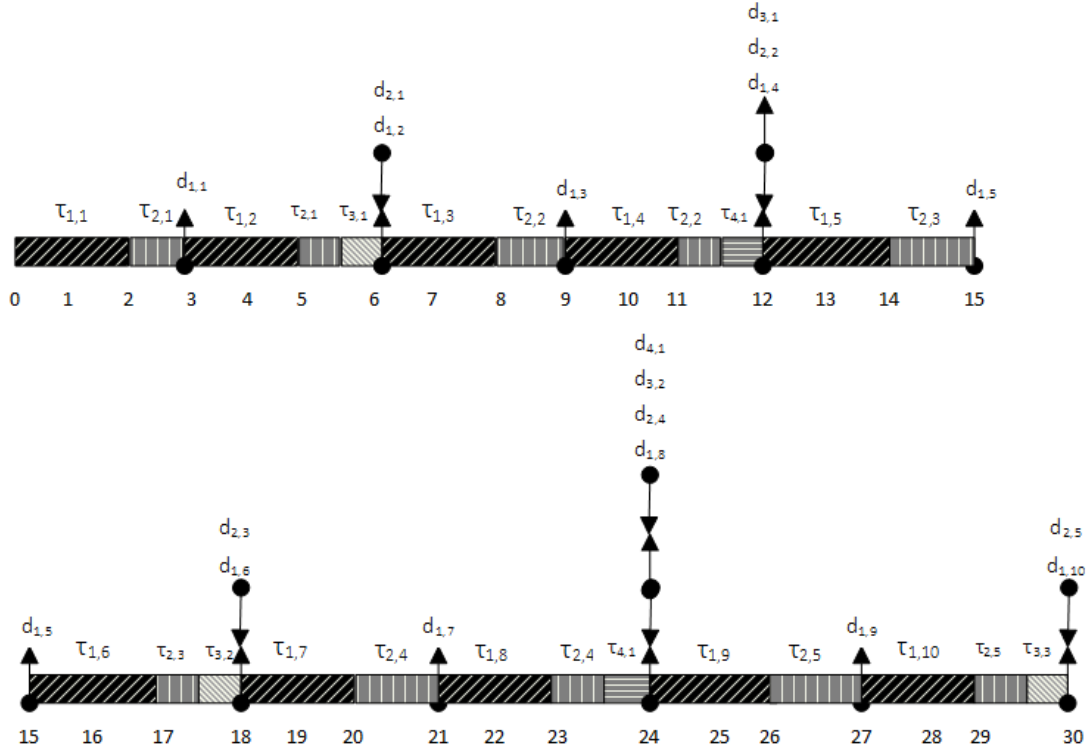


Figure 4.2 RM scheduling of  $\gamma$  in Example 1

**Theorem 2.** A group of real-time tasks  $\Upsilon_x = \{\text{task}_1, \text{task}_2, \dots, \text{task}_l\}$  is schedulable if all tasks in  $\Upsilon_x$  are schedulable.

**Theorem 3.** The batch of real-time tasks called periodic tasks set represented by  $T = \{\text{task}_1, \text{task}_2, \dots, \text{task}_n\}$  is deemed feasible if all tasks groups  $\Upsilon_1, \Upsilon_2, \dots, \Upsilon_x$  are schedulable.

#### 4.2.4 Cost Model

Scheduling decisions by integrating cost parameters change the way computational resources are selected to fulfill the user  $QoS$  criteria. The data-intensive real-time tasks are submitted to the broker which searches resources to process tasks within deadlines and user specified budget constraints. The feasibility of tasks groups on computational

resources is checked by considering data transfer time, transfer costs, computational cost, tasks deadlines, and computational power of the resources. The basic parameters considering for feasibility decisions in this research are:

1. user specified budget,
2. tasks deadlines

The resources which can execute tasks within deadlines in a minimum cost by considering all data and processing constraints are selected. By introducing cost model, Theorem 3 can be extended in Theorem 4 for checking schedulability of modified task set.

**Theorem 4.** The batch of real-time tasks called periodic tasks set represented by  $T = \{task_1, task_2, \dots, task_n\}$  is deemed feasible with minimum cost if all tasks groups  $Y_1, Y_2, \dots, Y_x$  are schedulable by following all constraints and holding inequality (4.12).

$$\text{cost}_T \leq \text{Budget} \quad (4.12)$$

where  $\text{cost}_T$  is the total cost incurred by the batch of tasks, and Budget is the total user specified budget. The cost of a resource can be expressed as execution cost per Million of Instructions (MI), processing cost per unit time, processing cost per task, or simulation cost per unit time etc. The cost for a single task is the sum of task execution cost and the data files transfer cost.

### 4.3 TIME AND COST-EFFICIENT SCHEDULING ALGORITHM

The Algorithm 1 determines the schedulability of real-time independent tasks set consisting of tasks with different data files and timing constraints. The execution procedure of the tasks involves checking tasks groups feasibility which cumulatively

constitutes tasks set. The  $m$  number of tasks in a group are checked on  $r$  number of distributed computing resources where  $r \gg m$ . Depending on the user budget and tasks scheduling preferences, the main objective of the algorithm is to execute distributed data-oriented applications by selecting computing resources such that the tasks are processed with minimum total execution time and cost while tasks deadlines are respected. The proposed algorithm works in three parts: (a) task initial feasibility checking which predicts task's basic feasibility within deadline by searching initial feasible computing resources, (b) task final feasibility and cost analysis which determines task's schedulability after considering all the associated constraints, and finally (c) task's dispatching to the best suitable resources after fulfilling all the prerequisites. The first two parts are the matching and mapping parts which create set of time- and cost-efficient computing-storage resources pairs. The third part is the dispatching part which ensures that the selected resources can process tasks within time and budget constraints. By cost we mean the sum of task's execution cost and data-files transfer cost. Similarly, the total execution time to minimize is the sum of tasks actual execution time and transfer time incurred by transferring data-files from the storage resources to the computing resources where the task is executed. The data-files are replicated on multiple storage resources and the resource which has minimum transfer cost is selected for data-file transfer. The computing resource capability for executing task is checked by analyzing task feasibility on *PNP* points. As a result, the compute resource that can execute task by maintaining the deadline is initially selected from the list of available computing resources. The selected resource is called as initially feasible resource. The service requests are provisioned according to the described scheduling strategy. The service request is provisioned such that the total execution time of the task set, and incurred cost is minimized. To ensure the fulfilment

of the aforementioned two objectives, the set of storage resources are demonstrated which accommodate data-files needed for the task  $task_i$  after identifying the initially feasible resources. A single file is assumed to be replicated on more than one storage resources, so the resource which has less transfer time and cost is selected. All such computing-storage resources pairs are further checked for calculating total execution time. The total execution time is the sum of all-time factors. If the total time is within the task  $task_i$  deadline and the total cost is within the user specified budget, the compute-storage resources pair is declared feasible for assigning  $task_i$ . After selecting all such pairs for all tasks in a group, the tasks are then dispatched to the qualified resources by the dispatcher and all required files are transferred. The tasks are scheduled, and computations are carried out. In this way, if all tasks in a group  $Y_x$  are scheduled, then the group  $Y_x$  is said schedulable by the Algorithm 1. Furthermore, if all groups are scheduled then the original task set  $T$  is declared schedulable with minimum time and cost. The resource allocation procedure completes when all the tasks are dispatched to the resources and the unmapped queue becomes empty. The pseudocode of the tasks mapping and dispatching procedures is given in Algorithm 1.

The purpose of Procedure 1 is to find the suitable compute-storage resource pairs for each data-file required by a task. For each task, set of required data-files and initially feasible computing resources are passed as input arguments to the file transfer time calculating function. For each data file, the storage resources are identified and the best data storage resource which qualifies the minimization criteria (transfer time and cost) are selected for retrieving data file. For each data file, all possible combinations of initially feasible compute-storage resource pairs are tried and finally, the right combination is returned with decreased transfer time and execution cost.



---

**Algorithm 1** Time and cost-efficient assignment of real-time data-intensive group of tasks to the HPC resources

---

**Input:** Computing resources sorted in descending order of processing capacities, and a group  $Y_x$  of unmapped real-time tasks ordered by RM priorities and having budget constraints.

**Output:** Time and cost-efficient real-time data-intensive tasks schedule on HPC resources.

**Procedure**

**for all**  $task_i \in Y_x$  **do**

    compute  $PNP_i = \{x.period_l | l = 1, \dots, i; x = 1, \dots, [period_i \setminus period_l]\}$ ;

**// Determining task initial feasibility**

**for all** available computing resources  $CR_r \in CR$  **do**

**for all**  $t \in PNP_i$  **do**

            calculate  $EET_{ir}$ ;  $\blacktriangleright$  gives minimum  $EET$  because resources are already sorted

**if**  $EET_{ir} \leq t$  **then**

$CR_i \leftarrow CR_r$ ;  $\blacktriangleright$   $CR_i$  is set of comp resources on which  $task_i$  is initially feasible

**break**;  $\blacktriangleright$  break if  $t_p$  is found

**end if**

**end for**

**end for**

**if**  $DF_i$  do not locally exist **then**

$CD_i \leftarrow FT(CR_i, DF_i)$ ;  $\blacktriangleright$   $CD_i$  is comp-storage resource pairs set for which  $DF_i$  has min transfer time and cost

**end if**

    calculate  $TT_{ir}$ ;  $\blacktriangleright$   $TT$  on  $CD_i$

**// Determining the task final schedulability and cost analysis**

**if**  $TT_{ir} \leq t$  **AND**  $cost_i \leq Budget$  **then**  $\blacktriangleright$  if cost of  $task_i$  is within the user budget

        mark  $CD_i$  feasible for  $task_i$ ;

**end if**

**end for**

**// Dispatching tasks to the feasible computing resources**

**for all** schedulable tasks  $task_i$  **do**

    submit  $task_i$  to  $CR_r$ ;

    transfer all required files to  $CR_r$ ;

    update resource information directory;

    remove  $task_i$  from unmapped tasks list;

**end for**

initialize computing resources to maximum processing powers and update resource information directory;

**end procedure**

---

**Procedure 1**  $FT(CR_i, DF_i)$

---

Specify  $DR_i$ ;  $\blacktriangleright$  storage resources on which files  $DF_i$  are stored

---

---

```

for all  $f_x \in DF_i$  do
  for all  $CR_z \in CR_i$  do      ▶ Compute resource  $z$  on which  $task_i$  is initially feasible
    for all  $dr_j \in DR_i$  do      ▶ Storage resource  $j$  where  $f_x$  is stored
       $C_{zj} \leftarrow (FT_{zj}, cost_x)$  ▶ transfer time and cost pair for file  $f_x$  in matrix  $C$ 
    end for
  end for
   $A_x \leftarrow (zdr, \min(C))$  ▶ pair of comp-storage resources for which transfer time and cost for  $f_x$  is min
end for
return  $(A)$ ; ▶ return comp-storage resources vector on which transfer time and cost for  $f_x$  is min

```

---

## 4.4 PERFORMANCE EVALUATION

This section discusses the experimental set-up, the input data, and performance metrics used to evaluate the proposed resource allocation technique.

### 4.4.1 Experimental Setup

The proposed RA technique and the existing counterparts were simulated using synthetic task sets. These experiments were carried out in MATLAB 2019 on Intel Core i5 processor, 2.50 GHz CPU and 8 GB RAM running on Microsoft Windows 10 platform. The reason for using MATLAB is that it provides multiprocessing environment for solving complex mathematical problems demanding powerful computations. The HPC systems are difficult to implement practically due to the lack of real-life experimentation environment and multiple domain administration problems which make it difficult to acquire stable configuration for evaluation. In addition, acquiring practical HPC environment is almost impossible due to the dynamic variations in the number of users and resources at a particular moment, their characteristics, limited access, and inconsistent network conditions over the public network (Venugopal & Buyya, 2008). In addition, effective evaluation needs the study of RA technique using different user inputs and varying resource conditions. Therefore, we have created the same HPC simulated environment by managing

predefined resource and network configurations such as number of computing and storage resources connected by network links of various bandwidths randomly assigned within the range {1024, 2048} MB.

The heterogeneity in the modeled simulation environment was carried out by randomly generated resource characteristics, network bandwidths connecting computing and storage resources, files sizes, tasks workloads, and number of files required for each task. The data files requirements for each task were also randomly assigned in the range  $\{x, y\}$  showing minimum and maximum values respectively where  $x$  is assumed 1. It means that a task can demand at least 1 and at most  $y$  number of files. The files sizes are fixed at 100 MB each. To model the data files distribution, each of the data file was replicated on more than one storage resources. In our experimentation, we assume that there exist maximum 5 copies of any data file on the storage resources. The storage resources were decoupled from the nodes where computing resources are deployed. The computing resources were initially equipped with the full processing capabilities randomly chosen within the range {10000, 40000} Millions of Instructions per Second (MIPS). The data files were replicated on multiple storage resources. The files required by a task are either pre-fetched or transferred during execution. When the data files fetched for some higher priority task on the same computing resource is used by the lower priority task or when the required file is locally available, the file transfer time is taken as zero. This technique exploits both temporal and spatial locality of data access. This file transfer incurs communication cost and time.

The periodic tasks parameters i.e., required execution time and periods are generated by using normal distribution function. The execution time  $e_i$  for each task <sub>$i$</sub>  was generated in the range  $\{a, b\}$  representing the best- and worst-case execution

times respectively. We have considered worst case execution time equal to the  $period_i$  for task<sub>*i*</sub> in order to ensure the tasks schedulability in any changing environment. Similarly, each task generates a job after interval of  $\{\alpha, \beta\}$  seconds where  $\alpha = 100$  and  $\beta = 10000$ . In our task model, the period of the task<sub>*i*</sub> is equal to its deadline i.e.,  $period_i = d_i$ . Initially, tasks were assigned RM priorities such that the task with high rate has higher priority, where  $rate = \frac{1}{period}$ . The tasks from the superset are grouped into subsets based on data files demands.

The tasks groups as well as tasks inside each group are sorted based on RM priority assignment technique. Each task in the group has respective computation requirements and is entitled to get computational resource no later than the deadline. The tasks in a group are scheduled on the HPC system and computing resources are allocated based on RM priorities. Each task generates multiple jobs. Each job is generated after interval of  $\{\alpha, \beta\}$  seconds. The experimentation was carried out by considering different number of computing and storage resources. The above discussed setting is subsumed in Table 4.1.

The computation time and cost of each job is summed into the computational requirements of each task. It is assumed that the communication cost of each job is minimal and is merged with the computation demand  $e_i$  of the task in our experimental setup. The data file is supplied to the task when it is requested from the storage resource and hence response time is zero. The transfer cost per unit size of the data file between data storage and computational resource was randomly generated between 1 and 5. In addition, the unit processing cost of the computing resources was generated between 5 and 50 depending upon the resource computing power. It is assumed that the file transfer within the same node incurs 0 transfer cost.

Table 4.1 Simulation parameters settings

Parameters	Values
Bandwidth	1024 ~ 2048 MB
Task data files demand	{x, y}
File size	100 MB
Computing resource capacity	10000 ~ 40000 MIPS
$e_i$	{a, b}
period <sub>i</sub>	100 ~ 10000
Data files transfer cost	{1, 5}
Computing resource unit processing cost	{5, 50}

#### 4.4.2 Performance Metrics

The proposed RA approach evaluates the HPC resource set for each task, and the overall objective is to minimize the total execution time and cost. The total execution time is the cumulative time consumed by the task set after assigning all tasks groups to the available computing resources. This time is also known as makespan and mathematically defined as follows.

$$\text{Makespan} = \max (TT_1, TT_2, \dots, TT_l) \quad (4.13)$$

where  $TT_j$  represents the total execution time of tasks group  $j$ .

Similarly, cost of a task set  $T$  is the overall cost incurred by all tasks groups.

Mathematically,

$$\text{Cost}_T = \max (\text{Cost}_1, \text{Cost}_2, \dots, \text{Cost}_l) \quad (4.14)$$

and

$$\text{Cost}_j = \sum_{i=1}^x \text{cost}_i$$

The  $\text{cost}_i$  is a combined cost incurred by a task processing on a computing resource and data files transfer taken by a  $\text{task}_i$  in a group  $j$ . The time and cost for tasks context switching is negligible in our experiments and hence not included in the objective function.

## 4.5 RESULTS AND DISCUSSION

In this section, we evaluate the performance of our proposed algorithm by comparing it with the two methodologies, RDTA (Martel, 2020) and Greedy.

The makespan and cost minimization behavior of the proposed and the aforementioned two techniques was checked for the randomly generated task sets consisting of 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000 tasks. The plots reported in this paper are the average values of 300 runs of all the task sets. According to the tasks grouping criteria discussed in section 3(iii), the task sets are grouped into 5, 7, 7, 4, 8, 5, 7, 9, 9, and 10 groups respectively. Each group accommodates different number of tasks based on applied grouping criteria. The tasks grouping details is given in Table 4.3. The experiments were performed by checking system behavior on different number of computing resources. The number of computing resources was randomly generated within the range {10, 100}. We assume that the data storage resource gives response immediately when a request is made by a task for data file access and hence the response time is ignored. The time delay in preparing the computing resource is also taken as zero because in our system, the computational resource is supposed to be ready for task execution as soon as the task arrives at that resource.

It was observed that for small task sets, a smaller number of computing resources was involved as compared to the larger task sets. It is also understandable that choosing proper number of computing resources can contribute in maintaining tasks deadlines. If a smaller number of computing resources is selected as compared to the large number of task sets, then it is likely that some tasks may not be RM feasible due to long waiting queues which is very crucial in real-time systems.

The main objective of this evaluation is to reduce the makespan and execution cost of the application while tasks deadlines are intact. Figure 4.3 (a) and (b) depicts the normalized values of the makespan. The variation in magnitude depends upon the total number of tasks per task set, number of data files demands, and the computation and deadline requirements of each task. The lower the makespan value, the better the performance of the RA scheme. The other performance measurement criterion is the execution cost minimization. From Figure 4.4 (a) and (b) it is evident that decrease in makespan results in reduced processing cost.

It is known from Figure 4.3 (a) and (b), that the proposed technique continues to make scheduling decision by analyzing tasks feasibility on searching PNP sets and checking each scheduling point until some positive point  $t_p$  is found. Although the size of PNP set for tasks group  $\Upsilon_x$  becomes large if the ratio between the periods of the first and the last task  $\left(\frac{\text{period}_n}{\text{period}_1}\right)$  in  $\Upsilon_x$  is large which consumes time because large number of inequalities are tested but this procedure enhances the chance of task feasibility because more positive-negative points becomes available for testing tasks schedulability. Furthermore, all the initial feasible computing resources are encountered and the resource having minimum cost for the task execution is selected for task processing. The RDTA approach merely deals with executing tasks within deadlines and hence does not consider the cost parameters which is considerably high in that case. In the case of Greedy technique, the graph is steeply higher because a feasible resource is selected at random without considering the low time and cost constraints. So, the resources with high computing power are selected when termed feasible.

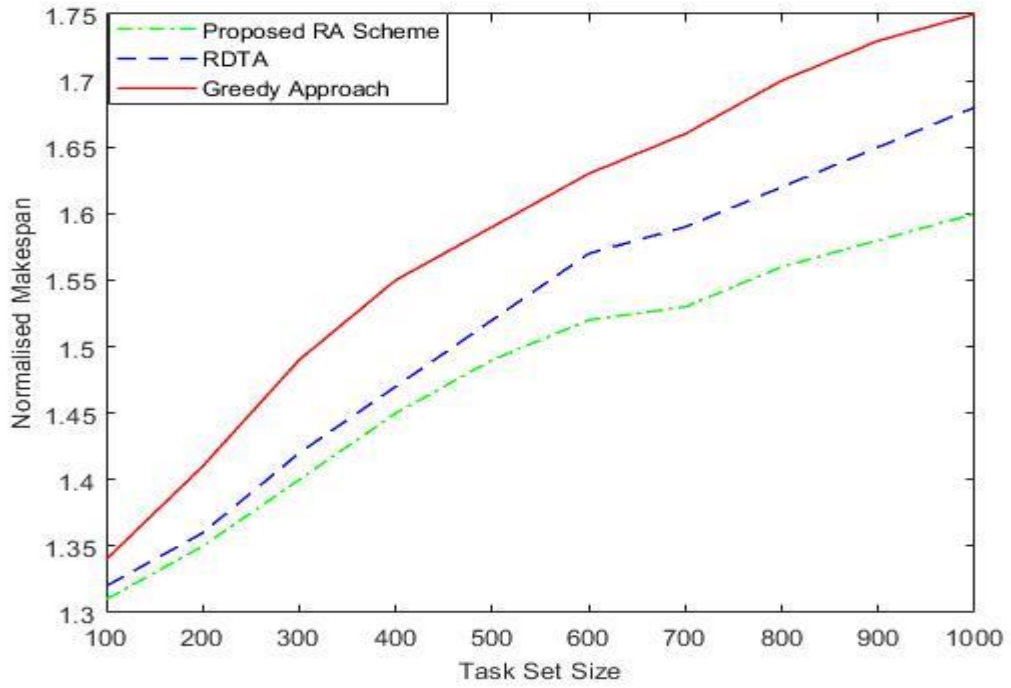
To further investigate the effectiveness of the proposed technique, we have conducted more experiments with different system settings. It is also noticeable from Figure 4.3 (a) and (b) that the time taken by all tasks test also increases uniformly as the number of tasks increase because more tasks are tested. It is obvious that the makespan of some task sets is high although the computing resources were operated at full speed because they need data files from remote storage resources which increase the total completion time. The resources when operate on full capacities consume high energy, but currently, energy efficiency is out of the scope of this research. The situations where makespan is low demonstrates that the data files are locally exist or perfected for some higher priority tasks and do not need re-fetching for the lower priority tasks which adds zero file transfer time to the overall execution time. The plots show that as the task set size increases, the makespan of the Greedy and RDTA grow as compared to the proposed approach. This growth in case of Greedy approach is because of making a greedy selection for the data storage and computing resources among multiple choices for data files accessing and task execution. This selection does not intelligently consider the minimization criteria. The RDTA mechanism also encounters high execution time and hence cost as shown in Figure 4.4 (a) and (b) because the data files replication is not taken into account when making a choice for data files fetch among storage resources. In the case of the proposed approach, the ratio  $\frac{\text{period}_n}{\text{period}_1}$  results in a larger value which constitutes a larger PNP set that provides more points for schedulability checking. This phenomenon provides more opportunities for task scheduling and hence results in large number of tasks meeting the deadlines constraints. Table 4.2 shows the formation of task groups in our



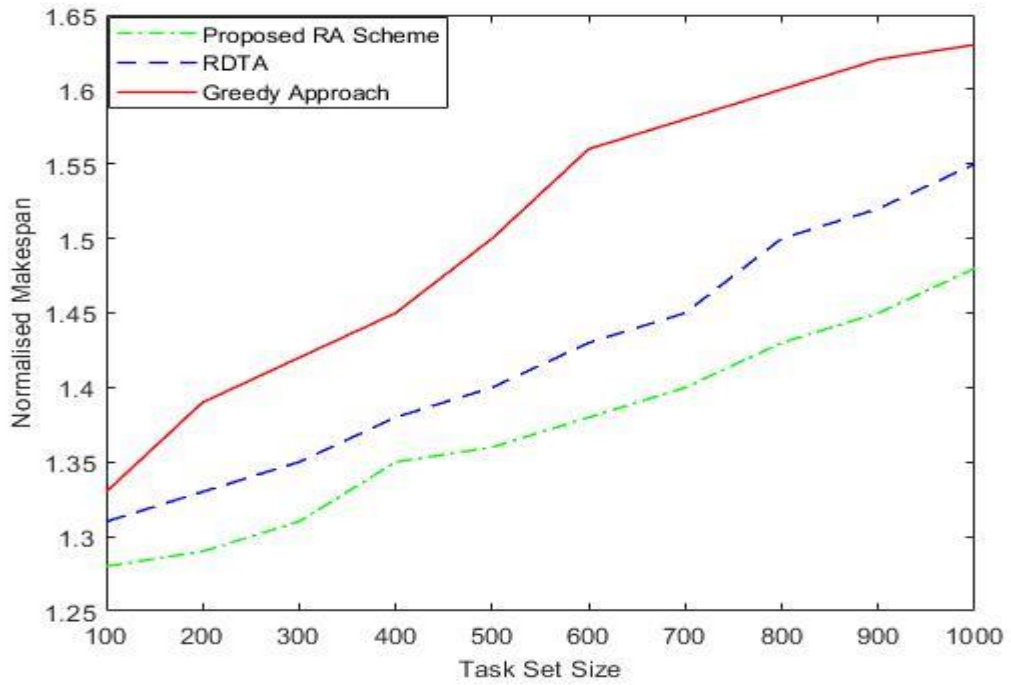
experimental evaluation based on randomly generated data files demand. The tasks groups are created as for as the inequality  $GU_{Y_x} \leq n(2^{1/n} - 1)$  in Theorem 1 holds.

Table 4.2 Tasks groups

<b>Task set size</b>	<b>Group (No. of tasks)</b>
100	TG1(25), TG2(10), TG3(30), TG4(8), TG5(27)
200	TG1(31), TG2(5), TG3(53), TG4(12), TG5(48), TG6(32), TG7(19)
300	TG1(4), TG2(64), TG3(20), TG4(25), TG5(40), TG6(126), TG7(21)
400	TG1(101), TG2(32), TG3(130), TG4(137)
500	TG1(10), TG2(23), TG3(116), TG4(67), TG5(50), TG6(39), TG7(120), TG8(75)
600	TG1(146), TG2(3), TG3(43), TG4(201), TG5(207)
700	TG1(2), TG2(133), TG3(108), TG4(7), TG5(211), TG6(120), TG7(119)
800	TG1(234), TG2(21), TG3(233), TG4(41), TG5(19), TG6(115), TG7(123), TG8(5), TG9(9)
900	TG1(112), TG2(21), TG3(34), TG4(321), TG5(232), TG6(18), TG7(116), TG8(29), TG9(17)
1000	TG1(12), TG2(109), TG3(120), TG4(32), TG5(19), TG6(129), TG7(127), TG8(245), TG9(21), TG10(186)

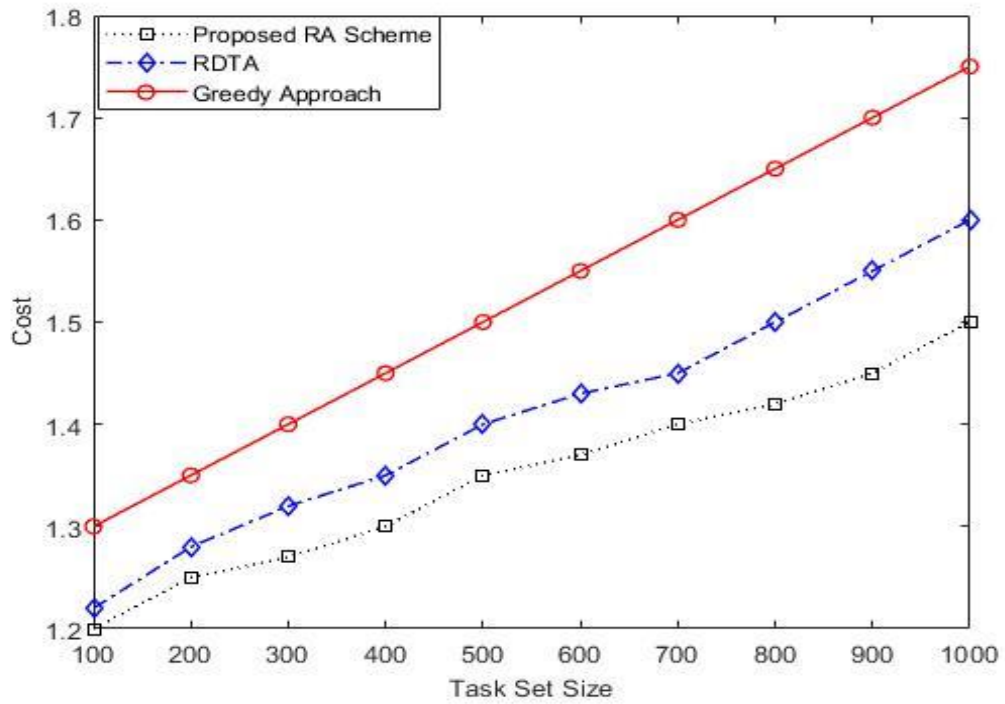


(a)

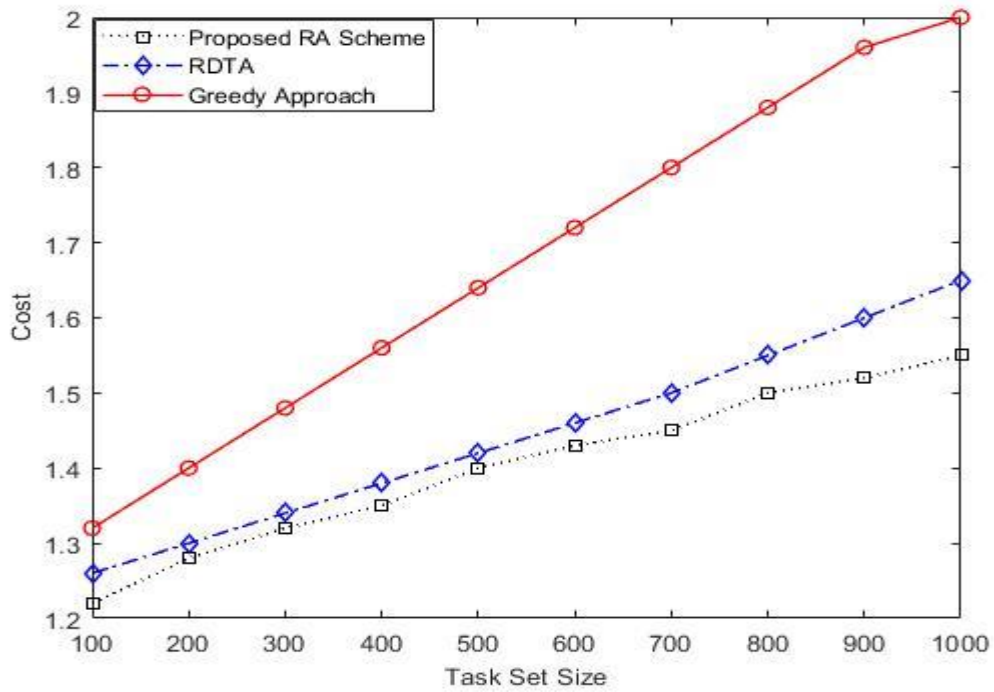


(b)

Figure 4.3 Average makespan



(a)



(b)

Figure 4.4 Average cost

#### **4.5.1 Effect of Data Files Transfer on Performance**

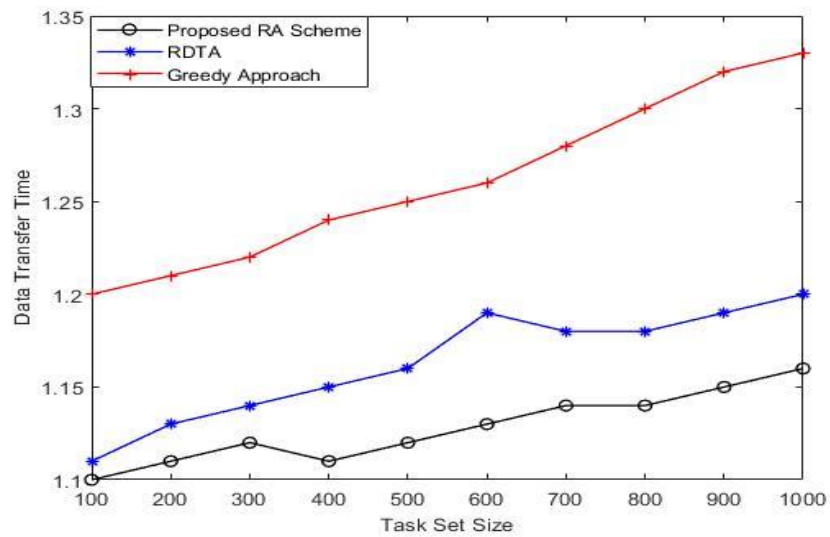
One of the basic components of calculating execution time is the data files transfer time incurred by transferring data files from the remotely located storage resources to the decoupled computing resources if the required files are not locally available. In addition to the makespan and cost values, the two more performance measures considered in the evaluation results are the percent share of the data transfer time and local data access.

In our experiments, the percent share of the data files transfer time in the makespan calculation is evaluated. The Figure 4.5 (a) and (b) plot the impact of the average data transfer time on the task sets. The lower value can put significant impact on reducing the overall makespan of the task set.

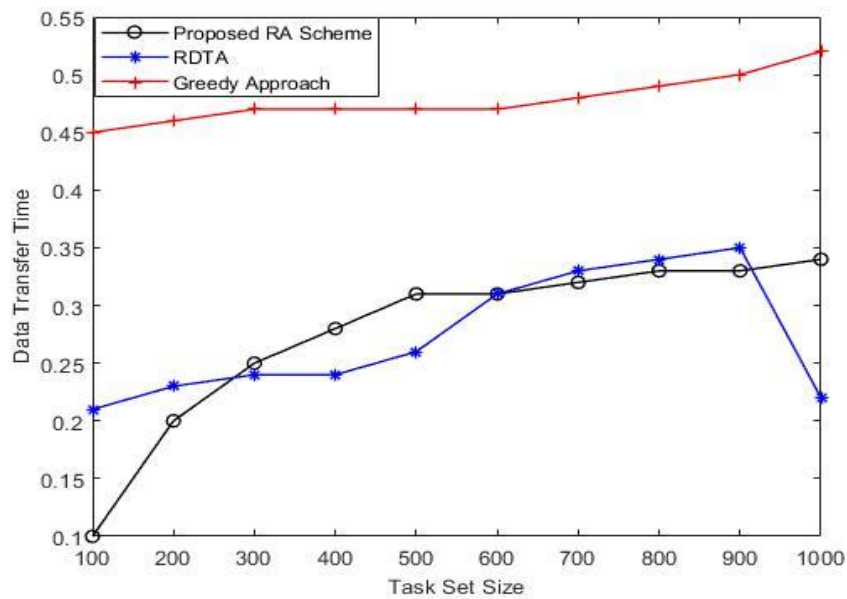
As it is known from the task workload that the lower priority tasks scheduled on the same computing resource can utilize the same data files retrieved for the higher priority tasks if the data requirements of the tasks are same. In that case, the data transfer time is zero. Additionally, the transfer time is also zero if the required file resides on the same node locally where the task is executing. In this case, the more locally accessed files decrease the impact of remote data files transfer on the performance. It is less likely that the task is scheduled on the same computing resource for which all the required data files are locally exist.

The above two factors can be correlated with the makespan calculation to indicate the impact of resource selection made by the RA scheme on achieving the decided objective. It is evident from Figure 4.5 that the Greedy and RDTA schemes do not intelligently adapt for the data files locality of access procedure and hence contribute to high data transfer percentage. The percentage of locality of access rises with the increase in the task set size. In comparison to the RDTA and Greedy

counterparts, there is a high chance for the lower priority tasks to reuse the pre-transferred data files by using the proposed RA scheme. In addition, it is more likely that the assigned tasks find the required data files locally. The Greedy approach exhibits degraded performance because there is a very less probability of finding appropriate computing resource for tasks assignment.



(a)



(b)

Figure 4.5 Data transfer time

#### 4.5.2 Impact on Resource Utilization

The utilization of the proposed RA scheme is measured on the basis of computing resources utilization in the HPC system. The resource utilization is directly related with the computation workload; when the task workload increases, the resource utilization also increases. The cumulative resource utilization can be calculated by the following equation.

$$Util_{cum} = \sum_{j=1}^r \frac{\text{tasks workload processing time by a resource}}{\text{resource active time}} \quad (4.15)$$

where  $Util_{cum}$  represents the cumulative utilization of all computational resources spent on processing tasks workload, and  $r$  represents the total number of computing resources engaged in processing tasks sets.

It is observed from Figure 4.6, that the proposed RA scheme improves the resource utilization by keeping resources as busy as possible. The resource utilization is lower for the tasks sets having a smaller number of tasks, but as soon as the number of tasks increases the resource utilization also increases. It means that the resource will be 100% utilized for the large task sets. This is an understandable phenomenon, because when the workload increases, more computational power is needed to complete tasks by their respective deadlines. If the computational power of the resources is relaxed for energy efficient allocation, then it is very likely that some of the tasks groups may not be feasible. Moreover, this also will pertain to an unfair comparison. When the number of tasks increases, the proposed procedure pushes the system power to grow rapidly in order to accommodate more tasks to maintain the deadline constraints. This behavior results in high energy consumption but in this research, we do not deal with the energy efficient perspective.

Figure 4.3 reveals that implementing the proposed approach, the minimum system utilization is between 70% and 72% for small task sets but touches 100% when the task computational demands increase. The maximum system utilization approaches 85% by the other counterparts.

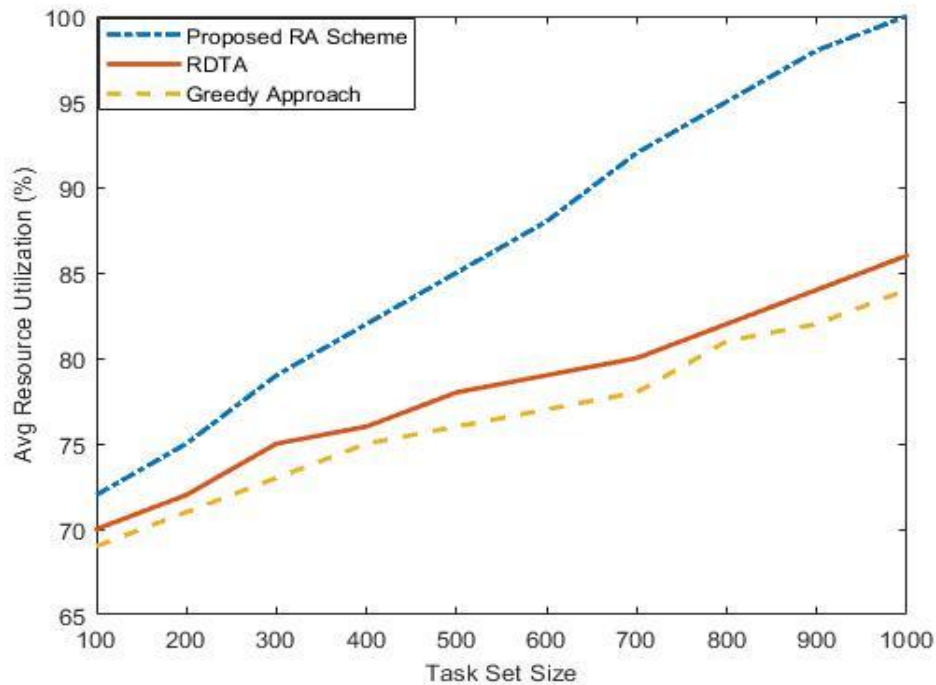


Figure 4.6 Effect on resource utilization

#### 4.6 CHAPTER SUMMARY

This chapter presented the problem of mapping a real-time application that requires data files that are replicated on multiple storage resources, to cloud computing resources. A task grouping technique was introduced that reduces the priority levels and execution time. The scheduling and resource allocation decision is driven by the need to improve the traditional performance parameters such as resource utilization and decrease the total application execution time and cost. In cloud computing

environment, the providers are incentivized by profit motives; while consumer would have a limited budget and would therefore, aim to execute the application at resources that provide service within the budget. In such environment, both provider and consumer aim to improve their utility. This research is user-centric which select computing and data storage resources in such way that makespan and cost is minimized while keeping the real-time deadlines. The results were obtained through mathematical formulations by modifying the original task model to incorporate the data files requirements. The proposed resource allocation scheme was compared with RDTA and Greedy approach and celebrated results were achieved.

As a future work, it will be interesting to rank the cloud resources based on different criteria such as resource computational powers, storage capacities, imbalance workloads, and cost and allocate them using machine learning algorithms.



## CHAPTER FIVE

### CONCLUSION AND FUTURE WORK

#### 5.1 CONCLUSION

In this thesis, we have discussed the real-time application scheduling and resource allocation problem in HPC environment. The resource allocation strategies in HPC paradigms (grid, cloud, fog, edge, and multicore) are studied in detail and working of each strategy is portrayed pictorially for easy understanding. The HPC strategies were particularly explored for applications with stringent timing constraints. The detailed analysis is provided on the basis of common parameters that are application type, operational environment, optimization goal, architecture, system size, resource type, optimality, simulation tool, comparison techniques, and input data. The taxonomies for different HPC systems show the structure of the systems and multi-objective criteria for scheduling real-time applications. From the literature study, the emergence of data-intensive real-time applications and their deployment on HPC systems motivated us to develop a resource scheduling and allocation strategy that predicts in advance the feasibility of such application on HPC resources. The results correctness of such applications not merely depend on the processing time on computational resources but the data files transfer which are needed for successful completion of the application. The proposed prediction-based model considers tasks multiple criteria such as resource ranking, tasks grouping, and user QoS parameters etc. for making scheduling decisions. In such RA scenario, some of the data files are transferred before the application starts execution on the computing resources while some are transferred during the execution. The tasks are prioritized for execution using a well-

known rate-monotonic scheduling algorithm. The prominent feature of this algorithm is its simple implementation and optimality. The proposed model is mathematically formulated.

We have also addressed problem of resource allocation for periodic real-time data-intensive tasks in cloud computing environment. In the proposed RA scheme, the tasks characteristics are modified by including the set of files needed by the task and the overall execution time and cost is minimized by grouping the tasks on the basis of common data-files demands. Additionally, the proposed technique considered the remotely located data storage resources on which the files are replicated. The best computing and storage resource pairs were selected for executing tasks based on user QoS criteria. The resource utilization perspective was also evaluated along with the other performance parameters. The achieved results by mathematical formulations verify the supremacy of the proposed RA scheme over the existing counter parts, RDTA and Greedy approaches.

## **5.2 THEORETICAL, PRACTICAL AND METHODOLOGICAL CONTRIBUTIONS**

The prime rationale of this research work was to explore and find limitations in the existing RA methodologies for deploying real-time applications having different optimization constraints. The other challenge was to propose a model to overcome the identified limitations. The main contributions are summarized as follows.

1. discussed and advanced the plethora of resource allocation algorithms for real-time applications by categorizing under two broad classes; distributed and non-distributed HPC systems. The distributed HPC systems include grid computing, cloud computing, fog computing, and edge computing

while non-distributed systems consist of multicore systems. The working of each mechanism is discussed in detail theoretically by identifying the successes and failures of each mechanism and presented pictorially which helps the common reader to understand the basic theme of each strategy. The evaluation and comparison on the basis of different performance parameters adds to the basic understanding which algorithm to adapt for different QoS criteria.

2. presented resource allocation prediction model on the basis of different task and resource constraints in the HPC environment that analyzes the feasibility of the real-time application before actually deploying on the computational resources. This model considers not only the timing constraints but also the data requirements of the data-intensive real-time application. The prediction analyzer takes into account different parameters like resource ranking, tasks grouping, positive negative points, and priority assignment.
3. identified the impact of investigating scheduling (positive) and non-scheduling (negative) points on task feasibility. This mechanism helps in reducing application completion time which in turn ensures fulfilling the deadlines.
4. specified a criterion for selecting computing and storage resources for real-time applications which need data-files for processing. The data-files are replicated on distributed data storage resources connected to the computing resources by network links. The proposed technique designs a criterion for the selection of storage and computing resources in a way

such that the resources are engaged for a short duration of time while respecting application deadlines.

5. developed a strategy for selecting cloud least cost computing and storage resources which can execute real-time application within deadlines and user budget constraints. The model considers both local and remote storage resources and helps in reducing data transfer time which ultimately reduces makespan. The mathematical modeling proves the validity of the proposed methodology.

### **5.3 FUTURE WORK**

The primary objective in executing real-time systems is to satisfy the deadlines which can be achieved by operating the HPC computational resources with full capacities. But this comes at the cost of high energy consumption. To cover this aspect, it is of interest to develop energy efficient mechanisms to execute real-time applications while respecting the deadlines. Another perspective of real-time system is the generation of infinite jobs while an HPC resource cannot be engaged forever. So, there is a need to extend the developed RA strategies to control number of jobs which can be completed within specified time on HPC resource.

Most of the existing literature on real-time system revolves around the dependent tasks with precedence constraints. In such systems the tasks execution is depending on the results of another tasks. But the real-time tasks cannot wait for a long time. So, the devised mechanisms can be modified to consider dependent tasks.

In another direction, the proposed scheme can consider tasks assignment to HPC resources using heuristic approaches to solve the unconstrained optimization

problems. The advantage of using heuristics is to accommodate applications with dynamic demands during execution.

## REFERENCES

- A. Beloglazov, J. Abawajy, R. Buyya. (2012). *Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing*, Future Generation Computer Systems, vol. 28, no. 5, pp.755–768.
- Abdullahi, M., & Ngadi, M. A. (2016). *Hybrid Symbiotic Organisms Search Optimization Algorithm for Scheduling of Tasks on Cloud Computing Environment*. Plos One, 11(6), pp.6-26.
- Amadeo M, Molinaro A, Paratore SY, et al. (2017). A Cloud of Things framework for smart home services based on Information Centric Networking. 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC). Calabria. pp.245-250.
- Amazon Elastic Compute Cloud (Amazon EC2). (2020). <http://aws.amazon.com/ec2/>. Retrieved on May 10, 2020.
- Amazon Elastic Compute Cloud (Amazon EC2). (2020). <http://aws.amazon.com/ec2/>. Retrieved on April 25, 2020.
- Amit Kumar Singh, Piotr Dziurzanski, Hashan Roshantha Mendis, Leandro Soares Indrusiak. (2017). *A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems*, ACM Computing Surveys (CSUR), Vol. 50 No 2, June 2017.
- Anwar, N., & Deng, H. (2018). *Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments*. Future internet, 10(1), pp.2-23.
- Awad, A., El-Hefnawy, N., & Abdel\_kader, H. (2015). *Enhanced Particle Swarm Optimization for Task Scheduling in Cloud Computing Environments*. Procedia Computer Science, 65, pp.920-929.
- B. A. Hridita, M. Irfan, M. S. Islam. (2016). Mobility aware task allocation for mobile cloud computing, *International Journal of Computer Applications*, Vol. 137, No. 9, 2016, pp.35-41.
- B. Yang, L. Guang, T. Santti, J. Plosila. (2013). *Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip*, *Microprocessors and Microsystems*, Vol. 37, No. 4–5, pp.460 – 471.
- Bini, E., & Buttazzo, G. C. (2005). *Measuring the Performance of Schedulability Tests*. Real-Time Systems, 30(1-2), pp.129-154.
- Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Freund, R. F. (2001). A Comparison of Eleven Static Heuristics for Mapping a

Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6), pp.810-837.

- C. L. Liu and J.W. Layland. (1973). Scheduling algorithms for multiprogramming in a Maintaining the Feasibility of Hard Real-Time Systems with a Reduced number of Priority Levels hard real-time environment. *Journal of the ACM*.20(1), pp.40–61.
- C. Marcon, E. Moreno, N. Calazans, F. Moraes. (2007). Evaluation of Algorithms for Low Energy Mapping onto NoCs," 2007 *IEEE International Symposium on Circuits and Systems, New Orleans, LA, 2007*, pp. 389-392.
- Caron, E., Chouhan, P., & Desprez, F. (2004). Deadline Scheduling with Priority for Client-Server Systems on the Grid. Fifth *IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, PA, 2004, pp. 410-414.
- Casanova, H., Berman, F., Obertelli, G., & Wolski, R. (2000). The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," SC '00: Proceedings of the 2000 *ACM/IEEE Conference on Supercomputing*, Dallas, TX, USA, pp. 60-60.
- Chaparro-Baquero, Gustavo A. (2018). *Memory-aware scheduling for fixed priority hard real-time computing systems*, FIU Electronic Theses and Dissertations, 3712, 2018.
- Chen, H., Zhu, X., Guo, H., Zhu, J., Qin, X., & Wu, J. (2015). Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. *Journal of Systems and Software*, 99, pp.20-35.
- Cloud Computing Taxonomy. (2019). URL: <http://agileanswer.blogspot.com/2010/12/cloud-computing-taxonomy.html>; accessed on February 25, 2020.
- D.W. Kim, K.-H. Kim, W. Jang, F. F. Chen. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing, *Robotics and Computer-Integrated Manufacturing*, 11th International Conference on Flexible Automation and Intelligent Manufacturing, vol. 18, no 3–4, pp. 223 – 231.
- Davidson, J. (2017). *Apple Homekit: The Beginner's Guide*. Van Helostein, 1st Ed. 2017. CreateSpace Independent Publishing Platform, USA.
- Deniziak, S., Ciopinski, L., Pawinski, G., Wieczorek, K., & Bak, S. (2014). Cost Optimization of Real-Time Cloud Applications Using Developmental Genetic Programming, 2014 *IEEE/ACM 7th International Conference on Utility and Cloud Computing*, London, 2014, pp.774-779.
- E. Ahmed et al. (2017). Bringing Computation Closer toward the User Network: Is Edge Computing the Solution? *IEEE Communications Magazine*, vol. 55, no. 11, pp. 138-144.
- F. Dong and S. G. Akl. (2006). Scheduling algorithms for grid computing: State of the art and open problems, Tech. Rep. Technical Report No. 2006-504, pp.1-55.

- Frederic, N.Z., & Yang, Y., (2017). *Effective Task Scheduling and Dynamic Resource Optimization based on Heuristic Algorithms in Cloud Computing Environment*, " *KSII Transactions on Internet and Information Systems*, vol. 11, no. 12, pp. 5780-5802.
- G. Sabin, M. Lang, P. Sadayappan. (2007). *Moldable parallel job scheduling using job efficiency: An iterative approach*, in *Job Scheduling Strategies for Parallel Processing* (E. Frachtenberg and U. Schwiegelshohn, eds.), vol. 4376 of Lecture Notes in Computer Science, pp. 94–114, Springer Berlin Heidelberg.
- G.L. Stavrinides, H.D. Karatza. (2017). *Scheduling real-time bag-of-tasks applications with approximate computations in SaaS clouds*, *Concurr. Comput.: Pract. Exper.*, 2017 e4208.
- G.L. Stavrinides, H.D. Karatza. (2018). *Scheduling data-intensive workloads in large scale distributed systems: trends and challenges*, in: *Modelling and Simulation in HPC and Cloud Systems, first ed.*, in: *Studies in Big Data*, vol. 36, Springer, pp.19–43.
- Gawali, M. B., & Shinde, S. K. (2018). Task scheduling and resource allocation in cloud computing using a heuristic approach. *Journal of Cloud Computing*, 7(1), pp.2-16.
- Georgios L. Stavrinides, and Helen D. Karatza. (2016). Stavrinides, G.L., & Karatza, H.D. (2016). Scheduling real-time parallel applications in SaaS clouds in the presence of transient software failures. 2016 *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pp.1-8.
- Georgios L. Stavrinides, Helen D. Karatza. (2015). A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds, In *Proceedings IEEE 3rd International Conference on Future Internet of Things and Cloud*, 24-26 Aug. 2015, Rome, Italy, pp. 231-239.
- Georgios L. Stavrinides, Helen D. Karatza. (2017). The impact of data locality on the performance of a SaaS cloud with real-time data-intensive applications," 2017 *IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Rome, 2017, pp.1-8.
- Georgios L. Stavrinides, Helen D. Karatza. (2018). *A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments*, *Multimedia Tools and Applications*, pp.1–17.
- Georgios L. Stavrinides, Helen D. Karatza. (2019). *An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations*, *Future Generation Computer Systems*, Vol. 96, 2019, pp. 216–226.
- Google Home Review. (2018). The smart speaker that answers almost any question. *The Guardian*. <https://www.theguardian.com/technology/2017/>



may/10/google-homesmart-speaker-review-voice-control. Retrieved on April 28, 2020.

- H. Alhussian, N. Zakaria, A. Patel, A. Jaradat, S. Jadid, A. Y. Ahmed, A. Alzahrani, S. Fageeri, A. Elsheikh, J. Watada. (2019). *Investigating the Schedulability of Periodic Real-Time Tasks in Virtualized Cloud Environment*, IEEE Access, vol. 7, pp.29533-29542.
- H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, A. Agarwal. (2011). *Sec: A framework for self-aware management of multicore resources*. Tech. Rep. Massachusetts Institute of Technology, 2011.
- H. Shojaei, A.-H. Ghamarian, T. Basten, M. Geilen, S. Stuijk, R. Hoes. (2009). A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management, in Proceedings of Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE, pp.917–922.
- Hameed Hussain, Muhammad Bilal Qureshi, Muhammad Shoab and Sadiq Shah. (2013). Load balancing through task shifting and task splitting strategies in multi-core environment, IEEE Eighth International Conference on Digital Information Management (ICDIM), 2013: pp. 385-390.
- Hengliang Tang, Chunlin Li, Jingpan Bai, JiangHang Tang, Youlong Luo. (2019). *Dynamic resource allocation strategy for latency-critical and computation-intensive applications in cloud-edge environment*, Computer Communications, Vol. 134, 2019, pp.70-82.
- Huangke Chen, Xiaomin Zhu, et al. (2015). Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. *Journal of Systems and Software*. 99. pp.20-35.
- Huangke Chen, Xiaomin Zhu, Hui Guo, Jianghan Zhu, Xiao Qin, Jianhong Wu. (2014). Reduced priority for real-time tasks under uncertain cloud computing environment, *The Journal of Systems & Software*, 2014.
- Hui Yan, Xiaomin Zhu, Huangke Chen, Hui Guo, Wen Zhou, Weidong Bao. (2019). *DEFT: Dynamic Fault-Tolerant Elastic Scheduling for Tasks with Uncertain Runtime in Cloud*. Information Sciences, 477, pp.30-46.
- Hussain, H., Malik, S. U., Hameed, A., Khan, S. U., Bickler, G., Min-Allah, N., Rayes, A. (2013). *A survey on resource allocation in high performance distributed computing systems*. Parallel Computing, 39(11), pp.709-736.
- I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, D. Soudris. (2013). Distributed run-time resource management for malleable applications on many-core platforms, in Proceedings of the 50th Annual Design Automation Conference, DAC '13, (New York, NY, USA), 168, pp.1–6, ACM, 2013.
- J. E. Boillat and P. G. Kropf. (1990). A fast-distributed mapping algorithm, in proceedings of the Joint International Conference on Vector and Parallel

Processing, CONPAR 90/VAPP IV, (London, UK, UK), Springer-Verlag, pp. 405–416.

J. W. S. Liu, Real Time Systems, Prentice Hall, 2000.

J. W. S. Liu. (2000). Real-Time Systems, Amazon Elastic Compute Cloud, 2000.

Javad Zarrin, Rui L. Aguiar, Joao Paulo Barrace. (2017). *Decentralized Resource Discovery and Management for Future Manycore Systems*, arXiv e-prints, arXiv:1710.03649, 2017.

Jiayin Li, Meikang Qiu, Zhong Ming, et al. (2012). Online optimization for scheduling preemptable tasks on IaaS cloud systems. *Journal of Parallel and Distributed Computing*. 72:666677.

Jyoti, A., Shrimali, M. (2020). *Dynamic provisioning of resources based on load balancing and service broker policy in cloud computing*. Cluster Computing, 23, pp. 377–395.

Kalaiselvi, S., Kanimozhi Selvi, C.S. (2020). *Hybrid Cloud Resource Provisioning (HCRP) Algorithm for Optimal Resource Allocation Using MKFCM and Bat Algorithm*. Wireless Pers Commun. 111, pp.1171–1185.

Kim, K. H., Beloglazov, A., & Buyya, R. (2011). *Power-aware provisioning of virtual machines for real-time Cloud services*. Concurrency and Computation: Practice and Experience, 23(13), pp.1491-1505.

Kołodziej, J., Khan, S. U., Wang, L., & Zomaya, A. Y. (2012). *Energy efficient genetic-based schedulers in computational grids*. Concurrency and Computation: Practice and Experience, 27(4), pp.809-829.

Kołodziej, J., Khan, S. U., Wang, L., Kisiel-Dorohinicki, M., Madani, S. A., Niewiadomska-Szynkiewicz, E., Xu, C. (2014). *Security, energy, and performance-aware resource allocation mechanisms for computational grids*. Future Generation Computer Systems, 31, pp.77-92.

Kumar, K., Feng, J., Nimmagadda, Y., & Lu, Y. (2011). Resource Allocation for Real-Time Tasks Using Cloud Computing, 2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN), Maui, HI, 2011, pp.1-7.

L. Schor, I. Bacivarov, D. Rai, H. Yang, S.H. Kang, L. Thiele. (2012). Scenario-based design flow for mapping streaming applications onto on-chip many-core systems, in Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES '12, (New York, NY, USA), ACM, pp.71–80.

Laplante, P.A. (2004). Frontmatter. In Real-Time Systems Design and Analysis. 3rd ed. Wiley.

- Lei Li, Quansheng Guan, Lianwen Jin, and Mian Guo. (2019). *Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queuing system*, *IEEE ACCESS*, Vol. 7, pp. 9912-9925.
- Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X., & Gu, Z. (2012). Online optimization for scheduling preemptable tasks on IaaS cloud systems. *Journal of Parallel and Distributed Computing*, 72(5), pp.666-677.
- Li, X., & Yin, M. (2013). A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem. *International Journal of Production Research*, 51(16), pp.4732-4754.
- Ligang He, Jarvis, S., Spooner, D., Xinuo Chen, & Nudd, G. (2004). Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids. Fifth IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, 2004, pp. 402-409
- Liu, S., Quan, G., & Ren, S. (2011). On-line scheduling of real-time services with profit and penalty. Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11. pp.1476–1481.
- Liu, Z., Qu, W., Liu, W., Li, Z., & Xu, Y. (2015). *Resource pre-processing and optimal task scheduling in cloud computing environments*. *Concurrency and Computation: Practice and Experience*, 27(13), pp.3461-3482.
- M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, et al. (2009). Above the clouds: A Berkeley view of cloud computing. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf>. Retrieved on May 05, 2020
- M. Malawski, G. Juve, E. Deelman, J. Nabrzyski. (2015). *Algorithms for cost and deadline constrained provisioning for scientific workflow ensembles in iaas clouds*,” *Future Generation Computer Systems*, vol. 48, pp.1–18.
- Mahmood, A., & Khan, S. A. (2017). *Hard real-time task scheduling in cloud computing using an adaptive genetic algorithm*. *Computers*, 6(2), pp.1-20.
- Martel, S. (2020). How cars have become rolling computers. <https://www.Theglobeandmail.com/globe-drive/how-cars-have-become-rollingcomputers/article29008154/>. Retrieved on May 02, 2020.
- Mckinsey. (2015). By 2025 Internet of Things Applications Could Have 11 Trillion Impact, <http://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internetof-thingsapplications-could-have-11-trillion-impact/>. Retrieved on April 28, 2020.
- Mehrin Rouhifar, and Reza Ravanmehr. (2015). A survey on scheduling approaches for hard real-time systems, *International Journal of Computer Applications*, Vol. 131, No. 17, 2015: pp. 41-48.
- Min Allah, Nasro & Khan, Samee. (2011). A hybrid test for faster feasibility analysis of periodic tasks. *International Journal of Innovative Computing*. 10. pp.1-10.

- Min-Allah N, Khan SU, Ghani N, Li J, Wang L, Bouvry P. (2012). *A comparative study of rate monotonic schedulability tests*. J Super Comput, 59(3), pp.1419–1430.
- Min-Allah, N., Qureshi M. B., Alrashed S., and Rana Omer F. (2019). *Cost efficient resource allocation for real-time tasks in embedded systems, Sustainable Cities and Society*, Elsevier, 48(2019), 101523, pp. 1-9.
- N.Mangla, M. Singh, and S. K. Rana. (2016). Resource scheduling in cloud environment: A survey, *Advances in Science and Technology Research Journal*, Vol. 10, No. 30, pp.38-50.
- Nadjaran Toosi, A., Sinnott, R., & Buyya, R. (2018). *Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka*. Future Generation Computer Systems, 79, pp.765-775.
- Naha et al. (2018). *Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions*, arXiv:1807.00976v1, Vol. X, 2018.
- Nasri M. (2017). *On flexible and robust parameter assignment for periodic real-time components*. ACM SIGBED Rev, 14(3), pp.8–15.
- Nasro Min-Allah, Hameed Hussain, Samee Ullah Khan, and Albert Y. Zomaya. (2012). Power efficient rate monotonic scheduling for multi-core systems, *Journal of Parallel and Distributed Computing*, Vol.72, No. 1, pp.48-57.
- Nasro Min-Allah. (2019). Effect of ordered set on feasibility analysis of static-priority system. *The Journal of Supercomputing*, 75(1): pp.475-487.
- P. Kokkinos, E. Varvarigos. (2009). *A framework for providing hard delay guarantees and user fairness in grid computing*, Future Generation Computer Systems, 25 (6), pp.674–686.
- Panda, S. K., Gupta, I., & Jana, P. K. (2015). *Allocation-aware Task Scheduling for Heterogeneous Multi-cloud Systems*. Procedia Computer Science, 50, pp.176-184.
- Pavan Kumar P., Satyanarayana C., Ananda Rao A., Radhika Raju P. (2019). *Empirical Evaluation of a Real-Time Task Priority Procedure on Small/Medium-Scale Multi-core Systems*. In: Krishna A., Srikantaiah K., Naveena C. (eds) *Integrated Intelligent Computing, Communication and Security*. Studies in Computational Intelligence, Vol. 771. Springer, Singapore.
- Q. Zhang, L. Gui, F. Hou, J. Chen, S. Zhu and F. Tian. (2020). Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud RAN. *IEEE Internet of Things Journal*. 7(4), pp.3282-3299.
- Qiushi Han, Tianyi Wang, and Gang Quan. (2015). Enhanced fault-tolerant fixed-priority scheduling of hard real-time tasks on multi-core platforms, *IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, Hong Kong, China, August 19-21, 2015: pp. 21-30.

- Qureshi, M. B., Alqahtani, M. A., & Min-Allah, N. (2017). *Grid Resource Allocation for Real-Time Data-Intensive Tasks*. IEEE Access, 5, pp.22724-22734.
- Qureshi, M. B., Alrashed, S., Min-Allah, N., Kołodziej, J., & Arabas, P. (2015). Maintaining the Feasibility of Hard Real-Time Systems with a Reduced Number of Priority Levels. *International Journal of Applied Mathematics and Computer Science*, 25(4), pp.709-722.
- Qureshi, M. B., Dehnavi, M. M., Min-Allah, N., Qureshi, M. S., Hussain, H., Rentifis, I., Zomaya, A. Y. (2014). Survey on Grid Resource Allocation Mechanisms. *Journal of Grid Computing*, 12(2), pp.399-441.
- R. L. Panigrahi, M.K. Senapaty. (2014). Real Time System for Software Engineering: An Overview, *Global Journal for Research Analysis*, Vol. 3, Issue 1, pp. 25-27.
- R. Mahmud, F. L. Koch, R. Buyya. (2018). Cloud-fog interoperability in iot-enabled healthcare solutions, in Proceedings of the 19th International Conference on Distributed Computing and Networking, ser. ICDCN '18. New York, NY, USA: ACM, 2018, pp.32:1–32:10.
- Raghavan, S., Sarwesh, P., Marimuthu, C., & Chandrasekaran, K. (2015). Bat algorithm for scheduling workflow applications in cloud," 2015 International Conference on Electronic Design, Computer Networks & Automated Verification (EDCAV), Shillong, 2015, pp.139-144.
- S. Abrishami, M. Naghibzadeh, D. H. J. Epema. (2013). *Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds*, Future Generation Computer Systems, vol. 29, no. 1, pp.158–169.
- S. Venugopal & R. Buyya. (2008). *An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids*. J. Parallel Distrib. Comput. 68(4), pp.471–487.
- Sait, S. M., Bala, A., & El-Maleh, A. H. (2015). *Cuckoo search-based resource optimization of datacenters*. Applied Intelligence, 44(3), pp.489-506.
- Sangwan, A., Kumar, G., & Gupta, S. (2016). *To Convalesce Task Scheduling in a Decentralized Cloud Computing Environment*. Review of Computer Engineering Research, 3(1), pp.25-34.
- Satish, K., & Reddy, A. R. (2018). Resource allocation in grid computing environment using genetic auction-based algorithm. *International Journal of Grid and High-Performance Computing*, 10(1), pp.1-15.
- Sha, L., Abdelzaher, T., Ārzén, K. E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., & Mok, A. K. (2004). *Real time scheduling theory: A historical perspective*. Real-Time Systems, 28(2-3 SPEC. ISS.), 101-155.

- Sindhu, S. (2015). Task Scheduling in Cloud Computing. *International Journal of Advanced Research in Computer Engineering & Technology*, 4, pp.3019-3023.
- Singh, V., Gupta, I., & Jana, P. K. (2018). *A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources*. *Future generation computer systems*, 79, pp.95-110.
- Statista. (2020). Number of internet of things (IoT) connected devices worldwide in 2018, 2025 and 2030. <https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/>. Retrieved on May 05, 2020.
- Sun, H., Yu, H., Fan, G., Liqiong Chen. (2020). *Energy and time efficient task offloading and resource allocation on the generic IoT-fog-cloud architecture*. *Peer-to-Peer Netw. Appl.* 13, pp. 548–563.
- Tarandeep Kaur, Inderveer Chana. (2016). *Energy aware scheduling of deadline-constrained tasks in cloud computing*, *Cluster Computing*, Vol. 19, No. 2, 2016, pp.679–698.
- The ambient. (2020). <https://www.the-ambient.com/guides/samsung-smartthings-guidesmart-home-163>. Retrieved on May 01, 2020.
- Tsai, C., Huang, W., Chiang, M., Chiang, M., & Yang, C. (2014). A Hyper-Heuristic Scheduling Algorithm for Cloud. *IEEE Transactions on Cloud Computing*, 2(2), pp.236-250.
- W. Shu, J. Cao, Q. Zhang, Y. Li, L. Xu. (2016). Edge computing: vision and challenges, *IEEE Internet of Things Journal*, vol. 3, no. 5, pp.637-646.
- Wang, J., Liu, A., Yan, T., & Zeng, Z. (2018). *A resource allocation model based on double-sided combinational auctions for transparent computing*. *Peer-to-Peer Networking and Applications*, 11(4), pp.679-696.
- Wu, X., Deng, M., Zhang, R., Zeng, B., & Zhou, S. (2013). *A Task Scheduling Algorithm based on QoS-Driven in Cloud Computing*. *Procedia Computer Science*, 17, pp.1162-1169.
- Xi, Sisu. (2014). *Real-time virtualization and cloud computing*, All Theses and Dissertations (ETDs). 1366, 2014.
- Xiaomin Zhu, Laurence T. Yang, Huangke Chen, Ji Wang, Shu Yin, and Xiaocheng Liu. (2014). *Real-Time Tasks Oriented Energy-Aware Scheduling in Virtualized Clouds*. *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp.168-180.
- Xie, T., & Qin, X. (2005). Enhancing security of real-time applications on grids through dynamic scheduling. In *Proceedings of the 11th international conference on Job Scheduling Strategies for Parallel Processing (JSSPP'05)*. Springer-Verlag, Berlin, Heidelberg, 219–237.

- Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, Shiyao Hu. (2019). *Scheduling in cloud using fuzzy dominance sort-based HEFT*, *Future Generation Computer Systems*, Vol. 93, 2019, pp.278-289.
- Xu, J., & Parnas, D. (1990). *Scheduling processes with release times, deadlines, precedence and exclusion relations*. *IEEE Transactions on Software Engineering*, 16(3), pp.360-369.
- Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, A. Keren. (2000). *An opportunity cost approach for job assignment in a scalable computing cluster*, *IEEE Transactions on Parallel and Distributed Systems*, 11 (7), pp.760–768.
- Y. Jiang, X. Shen, J. Chen, R. Tripathi. (2008). Analysis and approximation of optimal co-scheduling on chip multiprocessors, in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT '08*, (New York, NY, USA), ACM, pp. 220–229.
- Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief. (2017). *A survey on mobile edge computing: The communication perspective*, *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp.2322–2358, 2017.
- Zhang, Y., Tian, Y., Fidge, C., & Kelly, W. (2016). Data-aware task scheduling for all-to-all comparison problems in heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 93-94, pp.87-101.