Advance Planning, Reservation and Revenue Management in a Grid System

A thesis submitted during 2014 to the University of Hyderabad in partial fulfillment of a Ph.D degree in School of Computer and Information Sciences

By

Rusydi Umar

School of Computer & Information Sciences
University of Hyderabad
(P.O.) Central University, Gachibowli
Hyderabad - 500 046
Andhra Pradesh, India
September, 2014

To

The soul of my beloved father Prof. Umar Asasuddin, dip. TEFL, LEL, MA…

My beloved mother Mrs. Zaidar, B.A…

My beloved wife Dian Handayani, S.Si….

My beloved sons Dzikran and Zafran…

My dear Brother & Sisters...

C E R T I F I C A T E

This is to certify that the thesis work entitled **"Advance Planning, Reservation and Revenue Management in Grid System"** submitted by **Mr. Rusydi Umar** bearing (Regd. No. **07MCPC16**) in partial fulfillment of the requirements for the award of **Doctor of Philosophy in Computer Science** is bonafide work carried out by him under our supervision and guidance.

The thesis has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Prof. Arun Agarwal

(Supervisor)
School of Computer and
Information Sciences,
University of Hyderabad,
Hyderabad.

Prof. C. R. Rao

(Supervisor)
School of Computer and
Information Sciences,
University of Hyderabad,
Hyderabad.

Dean

School of Computer and
Information Sciences
University of Hyderabad,
Hyderabad.

DECLARATION

I, Rusydi Umar hereby declare that this thesis entitled **"Advance Planning, Reservation and Revenue Management in Grid System"** submitted by me under the guidance and supervision of **Prof. Arun Agarwal** and **Prof. C. Raghavendra Rao** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma.

Rusydi Umar

Date:

Signature of the Student:

Regd. No. 07MCPC16

ABSTRACT

In most grid systems with traditional scheduler, submitted jobs are placed in the wait queue if mandated resources are not available for them. Every grid system may use a different scheduling algorithm that executes jobs based on different parameters, such as number of resources, start time, duration of execution etc. With these scheduling algorithms, there is no guarantee about when these jobs will be executed.

Advance Planning and Reservation in a Grid System allows applications to request resources from multiple scheduling systems at a specific time in future and thus gain simultaneous access to sufficient resources for their execution. This thesis proposes a novel advance planning and reservation strategy namely First Come First Serve Ejecting Based Dynamic Scheduling (FCFS-EDS) to increase resources utilization in a grid system. The proposed strategy can handle both type of jobs which are parametric and MPI job. To achieve this, the thesis introduces a new data structure and a new notion that maps a user job from a virtual compute nodes (called logical view) to actual compute nodes (called physical view) at the time of execution. An FCFS-EDS's lemma ensures the success of such a mapping with increased resource utilization.

This thesis also studies how to increase incentives or profits for the resource providers. Toward fulfilling this study, the thesis suggests implementation of revenue management (RM) techniques to increase the revenue of resource providers.

Acknowledgements

I owe my deepest gratitude to my supervisors **Prof. Arun Agarwal** and **Prof. C. Raghavendra Rao** whose valuable guidance and support from preliminary to concluding level enabled me to develop an understanding of the subject. I am thankful to them for their patience and constant encouragement in motivating me with the words of hope throughout this work.

It is pleasure to thank DRC members, **Dr. Siba Kumar Udgata** and **Dr. Rajeev Wankar** for their valuable suggestions in completing my research work. I would like to thank **Prof. Robert L. Phillips** (Columbia University) for giving me a material to support my research work.

I would like to express my sincere thanks to Dean, School of Computer and Information Sciences **Prof. A. K. Pujari** for his cooperation.

I would like to thank Indian Council for Cultural Relation (ICCR) for providing the fellowship for the period of Dt: 17-08-2007 to 17-08-2012 to pursue this research in the field of grid computing. I am thankful to Director of ICCR Hyderabad, **Mrs. Laxmi**, and also to the officer of ICCR, Hyderabad, **Mrs. Uma** for their cooperation and support.

I would like to thank Rector of University of Ahmad Dahlan, Yogyakarta, Indonesia, **Dr. Kasiyarno, S.H., M.Hum.**, for giving me a chance and permission to pursue a PhD course in School of Computer and Information Sciences, University of Hyderabad, India. I would like to thank **Sri Winiarti, S.T., M.Cs.** for being Head in charge of Department of Informatics Engineering, University of Ahmad Dahlan, Yogyakarta, Indonesia, when I withdrew from the headship for pursuing PhD course in School of Computer and Information Sciences, University of Hyderabad, India.

I would like to avail this opportunity to thank my parents, the late **Prof. Umar Asasuddin, dip. TEFL, LEL, M.A.** and **Mrs. Zaidar, B.A.** whose good wishes enabled me to purse and achieve my goal. I gratefully thank my wife **Mrs. Dian Handayani, S.Si** and my two sons **Dzikran Azka Sajidan** and **Zafran Hulaif Rusydi** for their concern and moral support throughout my academics. I gratefully thank my parents in law, **Mr. Rahmadi** and **Mrs. Yuli Mahniar** for their concern and moral support.

I would also like to thank my brother **Mr. Muhammad Iqbal, S.T.** and my sisters **Mrs. Fairuz Rahmi, S.Si, M.A., Mrs. Hamidah, S.E, M.B.A.** and **Mrs. Suheir, S.Si.,** and their spouse, for encouraging me with their best wishes.

The warm support of all my friends in University namely, **Sreedhar Bhukya, Amer Ali Sallam, Mini, Akhter Mohiuddin, Rafah Mohammed, Mustafa Kaiiali, Mohammed Mansour Sholeh Saif, Abo Bakr** and other research scholars in the school, who enabled me to complete this thesis and have a wonderful time along the way.

Rusydi Umar

Contents

ABSTRACT v
Acknowledgements vi
List of Figures xi
List of Tables xvi
List of Algorithms xviii
1 Introduction 1
1.1 Grid Computing 1
1.1.1 Types of Grids 2
1.1.2 Advantages over Parallel and Distributed Computing 3
1.1.3 Grid Applications (Areas of Grid Computing Environment) 5
1.1.4 Resource Management in Grid Computing Environment 6
1.2 Motivation: Planning, Reservation and Revenue models 8
1.3 Contributions 9
1.4 Organization of the Thesis 10
2 Literature Review 11
2.1 Advance Reservation System 11
2.1.1 Rigid Reservation 12
2.1.2 Elastic Advance Reservation 13
2.1.3 Overlapping/Relax Advance Reservation 14
2.1.4 Flexible Advance Reservation (Static) 16
2.1.5 Flexible Advance Reservation (Dynamic, Physical View) 18
2.2 Data Structure 20
2.2.1 Segment Tree Data Structure 21
2.2.2 Calendar Queue Data Structure 24
2.2.3 Linked List Data Structure 26

2.2.4 GarQ Data Structure 28
2.3 Revenue Management System 30
2.4 Discussion and Summary 32
3 A Proposed Advanced Planning and Reservation Strategy 34
3.1 Flexible Advance Reservation: A new Perception 34
3.2 Types of Jobs 38
3.2.1 Parametric jobs 38
3.2.2 MPI jobs 38
3.3 FCFS-EDS Strategy 39
3.3.1 Model 39
3.3.2 Algorithm 40
3.3.3 Planning (Logical View) to Reservation (Physical View) Mapping 45
3.4 Illustration 50
3.5 Mapping of the Plan (Logical View) to Actual Compute Node (Physical View) 54
3.6 Discussion 56
4 Data Structure for FCFS-EDS Strategy 57
4.1 Introduction 57
4.2 Proposed Data Structure 58
4.2.1 Parametric Jobs 58
4.2.2 MPI Jobs 61
4.3 Planning Algorithm 64
4.3.1 Parametric Job 64
4.3.1.1 Searching an Available Virtual Node for Parametric job 64
4.3.1.2 Adding a Parametric Job 66
4.3.1.3 Deleting a Parametric Job 69
4.3.2 MPI Job 70
4.3.2.1 Searching an Available Virtual Nodes for MPI Job 70
4.3.2.2 Adding an MPI Job 71
4.3.2.3 Deleting an MPI Job 74
4.3.3 Complexity Analysis 75
4.4 Discussion 76
5 Revenue Management System 77

5.1 Introduction 77
5.2 Revenue Management System Strategy 78
5.2.1 Customers Segmentation and Price Differentiation 79
5.2.2 Capacity Allocation 80
5.2.2.1 Two Classes Capacity Allocation Problem 81
5.2.2.2 n Classes Capacity Allocation Problem (Heuristics approach) 85
5.3 Revenue Management System for Grid System 89
5.4 Example 93
5.5 Discussion 95
6 A New Grid System Architecture: Experiments and Results 97
6.1 A Proposed Architecture 97
6.2 Work Load Generator 100
6.3 Experiments and Results 101
6.4 FCFS-EDS for parametric job 102
6.5 FCFS-EDS for MPI job 107
6.6 Revenue Management System strategy for parametric job 113
6.7 Revenue Management System strategy for MPI job 120
7 Conclusions and Future Work 127
7.1 Conclusion 127
7.2 Future Work 129
List of Publication 130
References 131

List of Figures

Figure 2.1: Example of Bandwidth Utility Graph 22

Figure 2.2: Example of Advance Segment Tree data structure 22

Figure 2.3: Scheduling a new reservation on advance segment tree data structure . 23

Figure 2.4: Ten day calendar Queue 25

Figure 2.5: An example for the reserved bandwidth. The time ranges from 0 to 8 . 26

Figure 2.6: Data structure of a linked list corresponding to the reservation example shown in Figure 2.5 Each node contains the starting time, reserved bandwidth and a pointer to the next node 27

Figure 2.7: Example of reserved compute node in time space diagram with 5 compute nodes 28

Figure 2.8: User 6 arrives, request 2 compute nodes from time 6 up to time 8, defined by reserve(6; 8; 2) 29

Figure 2.9: A representation of storing reservations in GarQ with Sorted Queue and $\delta = 1$ 29

Figure 3.1: Flexible Advance Reservation 35

Figure 3.2: Planning Module 40

Figure 3.3: Planning (Logical View) to Reservation (Physical View) Mapping 45

Figure 3.4: Eleven users have been allocated using FCFS–EDS (Logical View).... 45

Figure 3.5: Actual Compute Node Mapping (physical view) at $t_0 = 18$ 46

Figure 3.6: Ten reservations have been allocated (Logical View) for parametric job 51

Figure 3.7: New user makes a request for parametric job 51

Figure 3.8: New user has been allocated using FCFS – EDS (Logical View) for parametric job 52
Figure 3.9: Allocation of ten MPI reservations in Logical View 53
Figure 3.10: MPI reservation request from a new user 53
Figure 3.11: A new user has been allocated using FCFS – EDS (Logical View) for MPI job 54
Figure 3.12: Allocation/Plan for reservations request using FCFS–EDS for parametric jobs (Logical View) 55
Figure 3.13: Actual Compute Node Mapping (physical view) at $t_0 = 18$ for parametric jobs 55
Figure 3.14: Allocation/Plan for reservation requests using FCFS–EDS for MPI jobs (Logical View) 55
Figure 3.15: Actual Compute Node Mapping (physical view) at $t_0 = 18$ for MPI jobs 56
Figure 4.1: Proposed Data Structure for Parametric Job 59
Figure 4.2: Resultant Data Structure for storing reservation requests of Table 4.1 . 61
Figure 4.3: Proposed Data Structure for MPI job 63
Figure 4 4: Resultant Data Structure for storing reservation requests of Table 4.2 . 64
Figure 5.1: We have 20 computing resources and we have set booking limit to b-1=9 82
Figure 5.2: Decision Tree of two-class capacity allocation problem 83
Figure 5.3: The relationship between booking limit and protection level 85
Figure 5.4: Booking periods for class 3, class2 and class 1 users 89
Figure 5.5: Booking limits for class 3, class2 and class 1 users 90
Figure 5.6: Revenue Management Module 90
Figure 6.1: An architecture for Advance Planning, Reservation and Revenue Management System 98
Figure 6.2: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #1) 103

Figure 6.3: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #2) 104
Figure 6.4: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #3) 104
Figure 6. 5: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #4) 105
Figure 6.6: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #5) 105
Figure 6.7: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #6) 106
Figure 6.8: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #7) 106
Figure 6.9: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #8) 107
Figure 6.10: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #1)..... 109
Figure 6.11: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #2)..... 109
Figure 6.12: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #3)..... 110

Figure 6.13: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #4)..... 110
Figure 6.14: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #5)..... 111
Figure 6.15: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #6)..... 111
Figure 6.16: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #7)..... 112
Figure 6.17: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #8)..... 112
Figure 6.18: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #1)..... 115
Figure 6.19: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking. (Experiment #2)..... 117
Figure 6.20: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking. (Experiment #3)..... 118

Figure 6.21: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #4) 119
Figure 6.22: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking. (Experiment #1) 122
Figure 6.23: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #2) 123
Figure 6.24: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #3) 124
Figure 6.25: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #4) 125

List of Tables

Table 3.1: Comparison between other scheduling strategies 37

Table 3.2: Reservation allocation at time slot t and $t + 1$ 48

Table 3.3: Reservation allocation at time slot t and $t + 1$ after $J(t + 1)$ multiplied by matrix C 49

Table 3.4: Parameters of parametric job reservation request 50

Table 3.5: Parameters of MPI job reservation request 52

Table 4.1: Parameters of reservation request..... 60

Table 4.2: Parameters of reservation request of MPI job 63

Table 4.3: Time asymptotic complexity 75

Table 5.1: Initial Booking Limit and Price for three classes user 93

Table 5.2: Mean and standar deviation..... 93

Table 5.3: Booking limit before and after update..... 95

Table 6.1: Experiments for parametric job 103

Table 6.2: Experiments for MPI job 108

Table 6.3: Market Segmentation and Price Discrimination..... 113

Table 6.4: Rate and Initial booking limit for the experiment 114

Table 6.5: Value of initial booking limit before and after update, for parametric job (experiment #1) 116

Table 6.6: Value of initial booking limit before and after update, for parametric job (experiment #2) 116

Table 6.7: Value of initial booking limit before and after update, for parametric job (experiment #3) 118

Table 6.8: Value of initial booking limit before and after update, for parametric job (experiment #2) 119
Table 6.9: Market Segmentation and Price Discrimination for MPI Job 120
Table 6.10: Rate and Initial booking limit for the experiment 122
Table 6.11: Value of initial booking limit before and after update, for MPI Job (Experiment #1) 123
Table 6.12: Value of initial booking limit before and after update for MPI Job (Experiment #2) 124
Table 6.13: Value of initial booking limit before and after update for MPI Job (Experiment #3) 125
Table 6.14: Value of initial booking limit before and after update for MPI Job (Experiment #4) 126

List of Algorithms

Algorithm 3.1: FCFS-EDS for papametric job 42
Algorithm 3.2: FCFS-EDS for MPI job 44
Algorithm 4.1: Searching available virtual node for parametric job 65
Algorithm 4.2: Adding a reservation for parametric job 68
Algorithm 4.3: Deleting a reservation for parametric job 69
Algorithm 4.4: Searching available virtual nodes for MPI job 70
Algorithm 4.5: Adding a reservation for MPI job 73
Algorithm 4.6: Deleting a reservation for MPI job 74
Algorithm 5.1: Update Booking Limit 92

1 Introduction

This chapter presents a high-level overview of the thesis and provides the motivation to propose a new advance reservation strategy and revenue-based resource management for Grid systems. This chapter also highlights contributions made and outlines the organization of this thesis.

1.1 Grid Computing

The Grid Computing discipline involves connections of a potentially unlimited number of ubiquitous computing devices within a network or grid and also involves the networking services. This new innovative approach to computing has similarities to a massively large power grid, such as what provides power to our business and homes daily. This innovative approach led to growing interest in coupling geographically distributed resources for solving compute intensive application and large-scale data intensive application. This approach is leading to what is popularly called peer-to-peer (P2P) computing networks [1] and the Grid [2]. The delivery of utility-based power has become second nature to many of us, worldwide. We know that by simply entering into a room and switching on the TV or any other electric devices, the power will be directed to the proper devices of our choice for that moment in time. In this same utility fashion, Grid computing openly seeks an infinite number

of computing devices into grid environment. Grid computing is able to add an infinite number of computing devices into any grid environment [3].

Grid [4][5] computing platforms enable the sharing, selection, and aggregation of geographically distributed heterogeneous resources (computers and data sources) belonging to different administrative organizations (also called Virtual Organizations (VOs)) for solving large-scale problems in science, engineering, and commerce.

1.1.1 Types of Grids

Depending on the purpose and target application domain, Grids can be classified into several types [6]:

- Computational grid. Computational grid provides distributed computing facilities for executing compute-intensive applications, such as Bag-of-Tasks (BoT) applications [7] where each application consists of independent tasks or jobs, and Monte Carlo simulations [8]. These applications scheduled on available resources by a project MyGrid [9], Nimrod-G [10], and SETI@home [11].

- Utility grid. Utility grid provides one or more grid services to end-users as information technology (IT) utilities on a pay-to-access basis. Utility grid has a framework for the negotiation and establishment of contracts. The allocation of resources based on user demands. Utility Data Center [12], at the enterprise level and Gridbus [13] at the global level are the example of utility grid.

- Data grid. Data grid provides the infrastructure to manage, transfer, and access large datasets stored in distributed repositories [14] [15]. In the area of astronomy [16], high energy physics [15], and climate simulation [17], there is a need for analysing large collections of data and sharing the results. These areas need requirements of scientific collaborations which are found in data grid. There are several projects involved in Data Grids, namely Avaki EII [18], Biogrid [19], Virtual Observatory [20], and LHCGrid [21].

- Knowledge grid. Knowledge grid works on data processing, data management, and knowledge acquisition. Knowledge grid also provides business analytics services driven by integrated data mining services. There are several projects in this involved in field of knowledge grid, namely the EU Data Mining Grid [22] and KnowledgeGrid [23].

- Application service provisioning grid. Application service provisioning grid provides an access to remote applications, modules, and libraries hosted on data centres or Computational Grids, e.g. NetSolve [24].

- Interaction grid. Interaction grid provides services and platforms for users to interact with each other in a real-time environment, e.g. AccessGrid [25]. Interaction grid is suitable for multimedia applications and those that require fast networks, for example video conferencing,

1.1.2 Advantages over Parallel and Distributed Computing

The advantages gained from grid computing can translate into competitive advantages in the marketplace. For example, the potential exists for grids to [26] [27]:

- Exploits underutilized resource. There are large amounts of underutilized computing resources in most organization. Most desktop machines are busy less than 5 percent of the time over a business day. Even in some organizations, the server machines can often be relatively idle. Grid computing provides a framework for exploiting these underutilized resources which in turn significantly increasing the efficiency of resource usage.

- Enables sharing of resources. Grid computing provides a framework to enable an organization to share their underutilized resources, which in turn increase their resources utilization factor.

- Transparent access to remote resources.

- Parallel CPU capacity. A compute-intensive grid application can be divided into a lot of smaller sub jobs. Each sub job can be executed on a different machine in the grid. To the extent that these sub jobs do not need to communicate with each other, the more scalable the application becomes.

- On demand aggregation of resources at multiple sites. A compute-intensive grid application which needs a lot of resources can aggregate the resources from multiple sites in the grid environment.

- Virtual resources and virtual organizations for collaboration. Grid computing provides an environment for collaboration among a wider audience. This can be done by offering important standards that enable heterogeneous systems to work together to form the image of a large virtual computing system. The resource providers and users of the grid can be members of several real and virtual organizations. A virtual organization (VO) is defined as a dynamic group of organizations, groups and/or individuals, who define the conditions, business objectives, and policies for sharing resources in the grid system.

- Reduced execution time of large-scale data processing applications. A data grid can expand data capabilities. Files or databases can span many systems and thus have larger capacities than on any single system. Such spanning can improve data transfer rates through the use of striping techniques which in turn reduce the execution time of large-scale data processing applications.

- Access to additional resources. The grid system can enable user to access additional resources such as, software and special devices. By accessing software with an expensive license, the grid is more exploiting software licence.

- Resource balancing. For a compute-intensive grid application, the grid can offer a resource balancing effect by executing jobs on a low utilization machines. The grid helps in taking advantage of time zone and random

diversity: grid users who are in peak-time hours can access resources that are in off-peak time where the load is likely less.

- Reliability. The grid system composed from aggregated resources form geographically different locations. With this condition, if there is a power or other kind of failure at one location, the other parts of the grid are not likely to be affected. When a failure is detected, grid system can automatically resubmit jobs to other machines on the grid.

- The grid user sometime has an unforeseen emergency demands. In this case the grid users do not have to owning the resources but they can rent external resources on the grid.

1.1.3 Grid Applications (Areas of Grid Computing Environment)

Some applications of grid computing, particularly in the scientific and engineering arenas, include, but are not limited to, the following [4]:

- Distributed supercomputing/computational science

- High-capacity/throughput computing: large-scale simulation/chip design and parameter studies

- Content sharing, for example, sharing digital content among peers (e.g., Napster)

- Remote software access/renting services: ASPs and web services

- Data-intensive computing: drug design, particle physics, stock prediction

- On-demand, real-time computing: medical instrumentation and mission-critical initiatives

- Collaborative computing (e-science, e-engineering): collaborative design, data exploration, education, e-learning

- Utility computing/service-oriented computing: new computing paradigm, new applications, new industries, and new business

1.1.4 Resource Management in Grid Computing Environment

The main components of a grid infrastructure are security, resource management, information services, and data management [28]. The term resource management in grid computing can be defined as operations that control the way that grid capabilities are made available for other entities like users, applications and services [29]. The goal of resource management is to ensure efficient utilization of computer resources and for optimization performance of specific tasks [30].

In global grid environment, users interact with a grid resource broker (GRB) that hides the complexity of resource management and scheduling. For the operation of a computational grid, the grid resource broker discovers properties of resources that the user can access using Grid Information Services (GIS), negotiates with (grid-enabled) resources or their agents using middleware services, maps tasks to resources (scheduling), stages the application and data for processing (deployment) and finally gathers results [31].

There are three types of resource management architecture approach which are centralized, decentralized, and hierarchical. Hierarchical and decentralized approaches are suitable for grid resource and operational management, because it is impossible to define an acceptable system-wide performance matrix and common fabric management policy [32].

To handle the wide variances in the software applications and hardware used in grid environments over different forms of grid networks, a software known as middleware is used. One of the most important components of the middleware is the resource manager which handles resource selection and job scheduling [33] [34].

There are two types of resource management which are global resource management and local resource management. Local resource management deals with

scheduling and managing resource at a particular site or resource provider. Like local resource manager, global resource manager does not own the resources at a site and therefore does not have control over them. The global resource manager must make best-effort decisions and then submit the job to the selected resources [35].

Grid resource management has a several different layers of schedulers. At the highest level are global resource management that may have a more general view of the resources but are very far away from the resources where finally the application will be executed. At the lowest level is a local resource management system that manages a specific resource or set of resources. Other layers may exist in between these resource managements, for example one to handle a set of resources specific to a project [36].

In local resource management, resources are accessed, assigned, and allocated according to Quality of Service (QoS) criteria, such as advance reservation, deadline and cost.

Therefore, we need to address the following questions, to use Grid resources:

1. How to know where the resources are?
2. How to identify resources?
3. How to get permission to use them?
4. How to know what are the applications available in these resources?
5. How to use the resources?
6. How to reserve resources for remote jobs?
7. How to get access to resources to all the machines simultaneously?
8. What happens if a resource fails?
9. How input/output files are managed?
10. What is a Revenue model for resource utilization?

To address questions like (6) and (10) requires an efficient planning, reservation and revenue management system which is dealt in the next section.

1.2 Motivation: Planning, Reservation and Revenue models

In most grid systems with traditional scheduler, submitted jobs are placed in the wait queue if mandated resources are not available for them [37]. Every grid system may use a different scheduling algorithm, for example First Come First Serve (FCFS), Shortest Job First (SJF), Earliest Deadline First (EDF), or EASY Backfilling [38] that executes jobs based on different parameters, such as number of resources, submission time, and duration of execution. With these scheduling algorithms, there is no guarantee about when these jobs will be executed [39].

To address the unwarranted waiting time issue and ensure that the specified resources are available for applications at a particular time in the future, we need an advance planning and reservation system [40]. Advance reservations [41] allow a user to request resources from multiple scheduling systems at a specific time in the future and thus gain simultaneous access to enough resources for their applications.

Buyya et al. [42] have introduced a Grid economy concept. This concept provides a mechanism for regulating supply and demand, and calculates pricing policies based on these criteria. Grid economy concept offers an incentive for resource owners to join the Grid, and encourages users to utilize resources optimally and effectively. A simple pricing model, used to determine the usage cost of each reservation (incentive for resource owner), has been provided by [43] [44] [45]. In simple pricing model, grid system only has one price for all customers. In this case, some compute nodes may become idle because of no reservation for them. The idea of revenue management is giving those idle compute nodes to the early reservations with a discount price. To increase incentives or profits, resources provider might need to adopt a more complex pricing model. In order to address this problem in this thesis we incorporate revenue management (RM) techniques to increase the revenue of resource provider.

1.3 Contributions

This thesis presents the following contributions made in advance planning, reservation and revenue-based resource management for Grid systems:

1. This thesis has attempted to broadly categorize these advance reservation strategies reported in the literature as: Rigid reservation, Elastic Advance Reservation, Overlapping/Relax Advance Reservation, Flexible Advance Reservation (Static), Flexible Advance Reservation (Dynamic, Physical View).

2. This thesis has proposed a novel advance planning and reservation strategy namely First Come First Serve Ejecting Based Dynamic Scheduling (FCFS-EDS), which falls into Flexible Advance Reservation (Dynamic, Logical View), to increase resources utilization in a grid system. In the earlier scheduling like Flexible Advance Reservation (Dynamic, Physical View), necessary physical resources were reserved for the job and user was notified accordingly. If the physical resources were re-assigned a fresh notification had to be sent. In FCFS-EDS, it schedules the job at a logical level and if accepted the user is notified only once. Finally this thesis proposed a definition for flexible advance reservation: A new perception.

3. This thesis has compared the parameters of the proposed scheduling strategy FCFS-EDS with advance reservation scheduling strategies that have been used by other researchers.

4. This thesis has proposed an FCFS-EDS's lemma to ensure "If there is a plan for scheduling a job on the consecutive time slot on virtual compute nodes (which are selected freely) (Logical View) then it guarantees that the job will be executed on a dedicated physical node for the required execution time (Physical View)".

5. This thesis has proposed a data structure, which falls into time slotted data structure to efficiently administer the proposed scheduling strategy.

6. To demonstrate the efficacy of the proposed FCFS-EDS strategy and the data structure, a novel simulator has been designed and implemented. This simulator has a potential to accommodate reservation for both Parametric as well as MPI job types.

7. This thesis has also proposed architecture of our work as a complete system. It shows the interaction between different components in the model to perform the proposed advance reservation scheduling strategy (FCFS-EDS) by incorporating revenue management in order to increase revenue/throughput.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 presents the literature review regarding advance reservation including its data structure. This chapter also presents literature review regarding revenue management in grid system.

Chapter 3 proposes advance planning and reservation strategy. This chapter introduces a definition for flexible advance reservation. The proposed advance planning and reservation strategy called First Come First Serve Ejecting base Dynamic Scheduling (FCFS-EDS) is done in logical view. This chapter introduces a new lemma (FCFS-EDS's Lemma) to ensure that a plan in logical view can always be mapped into physical view for execution.

Chapter 4 proposes the data structure for the proposed advance planning and reservation strategy. Chapter 5, presents the revenue management model that is incorporated in our scheduling strategy to increase revenue. Chapter 6 presents the architecture of our work as a complete system and the experimental result of our proposed advance planning and reservation scheme. We also present the results after incorporating revenue management system in it. Chapter 7 presents conclusions and provides directions for future work.

2 Literature Review

Advance reservation is the process of requesting resources for use at specific times in the future [41]. Common resources that can be reserved or requested are compute nodes, network bandwidth, storage elements or a combination of any of those. This chapter describes recent works to give an insight into the latest research advancements related to advance reservation in Grid Systems. The objective of Revenue Management System is to maximize the revenue by providing the right price for every product to different customer, and periodically updates the prices in response to market demand [46]. This chapter also describes recent works related to revenue management system in the Grid.

2.1 Advance Reservation System

In most grid systems with traditional scheduler, submitted jobs are placed in the wait queue if mandated resources are not available for them. Every grid system may use a different scheduling algorithm, for example First Come First Serve (FCFS), Shortest Job First (SJF), Earliest Deadline First (EDF), or EASY Backfilling [38] that executes jobs based on different parameters, such as number of resources, submission time, and duration of execution. With these scheduling algorithms, there is no guarantee about when these jobs will be executed [39].

To address the unwarranted waiting time issue and ensure that the specified

resources are available for applications at a particular time in the future, we need an advance planning and reservation system [40]. Advance reservations [41] allow a user to request resources from multiple scheduling systems at a specific time in the future and thus gain simultaneous access to enough resources for their applications.

Many earlier literatures discuss about advance reservation strategy to increase resource utilization. We have attempted to broadly categorize these advance reservation strategies reported in the literature as:

1. Rigid Reservation
2. Elastic Advance Reservation
3. Overlapping/Relax Advance Reservation
4. Flexible Advance Reservation (Static)
5. Flexible Advance Reservation (Dynamic, Physical View).

2.1.1 Rigid Reservation

When users request for compute nodes to execute their jobs, they must provide three parameters, start time, execution time and number of compute nodes [40]. The reservation system then searches for the availability of requested compute nodes in the specified time interval. If required nodes are unavailable, the request is rejected. This mechanism is known as rigid reservation and has been adopted into the Globus Architecture for Reservation and Allocation (GARA) [47] [48].

As a consequences of this mechanism (rigid reservation), if a request from a user is rejected due to the unavailability of compute nodes at the specific time as requested by a user, he or she may resubmit a new request with a modified parameter, such as different start time and/or execution time until available resources can be found. If this happens frequently, the scheduler works harder dealing with the same user request because his/her previous request was rejected. Ultimately, when the compute nodes are found, may be it is not a good one since it looks on the first available compute node. It will cause fragmentation, which is idle compute nodes between jobs; hence the utilization of the compute nodes goes down.

2.1.2 Elastic Advance Reservation

With rigid reservations, there is a shortcoming. If the users still want to reserve compute nodes then they must change the job parameters and again query for reservation until there is a match between the availability of compute nodes and the user's request. This mechanism brings an overhead due to communication traffic. Elastic reservation proposed by Sulistio et al. [43] takes the user request parameters as soft constrain. The reservation system instead of rejecting the request provides alternatives that can be selected by the user. This approach gives flexibility to users to select the best option or self-select in reserving their jobs according to their Quality of Service (QoS) needs, such as deadline.

In elastic reservation, to increase the probability of getting accepted, the user can query in interval time slots. This query operation has three parameters which are $ti_s, ti_e, dur?$, and $numCN?$, where ti_s is the earliest start time interval, ti_e is the latest end time interval, dur is the reservation duration time, and $numCN$ is compute node needed respectively. They have given the "?" sign indicates that this attribute is optional. They assume that $(ti_s - ti_e) \geq dur$.

When receiving such query, elastic reservation will try to find available compute nodes at ti_e, if there are no available compute nodes at time slot ti_e, then elastic reservation will give the alternatives that are available within the time interval given by user. Users can select their best option according to their Quality of Service (QoS) needs, such as deadline. Once a user selects one option of the given alternatives, then a user sends again his/her query. But, this time a user sends the query like the way in the rigid reservation, because there is a guarantee that computes node will be available for him/her.

Sulistio et al. [43] in finding the alternatives of available compute nodes to the users, they have compared between rigid reservation (RR), First Fit (FF) algorithm and their on-line strip packing (OSP) algorithm. Results show that their on-line strip packing (OSP) method performs better than first fit alternative (FF) and FF performs better than rigid reservation (RR). This strategy has been adopted in GridSim [49].

2.1.3 Overlapping/Relax Advance Reservation

Smith et al. [41] has introduced the impact of advance reservation in the scheduling system. They concluded that by adding advance reservation in the scheduler will increase the mean waiting time of query in the queue (non-reservation query) by 9% with backfilling [38]. This is done by selecting 10% of query as reservation query and 90% is non-reservation query, across different workload model. If the reservation query is increased to 20%, then the mean waiting time of the non-reservation query will increase by 37%. They also found by applying advance reservation in scheduling system will decrease the resource utilization between 54 and 59%. This is because of fragmentations or idle time gaps caused by advance reservation.

To overcome the problem of decreasing resource utilization caused by advance reservation Peng Xiao et al. [50] used the overlapped time slot table, because they believed that applications tend to overestimate the reservation deadline to ensure their completion [51] [52]. In this strategy, user jobs are scheduled even if there are reservation violations in the deadlines because of overlapping of jobs. They assumed that the real workload tends to overestimate the relative deadline with mean value by 35%. They have two parameter to minimize the probability of reservation violation, first p^*, the higher value of p^* indicates that RM (Resource manager) is conservative; a lower p^* shows that it is willing to take more risks to improve utilization of its resources. For example $p^* = 0.8$, it means that RM will accept the reservation if the probability of reservation violation is less than 20%. Second is v^*, is the threshold, by which RM decides whether a free time slot can be allocated to an overlapped reservation or not. So, v^* is a strategic parameter of RM, a higher value of v^* indicates that RM is conservative.

The increasing of reservation violation is the price they have to pay while using relaxed strategy, experimental results indicate that they can limit the violation rate in a relative low level by adjusting parameter v^*.

Experimental results show that this strategy can bring about remarkably higher resource utilization and lower rejection rate at the price of slight increase in reservation violation.

Chunming et al. [53] introduced resource capacity, i.e., a special kind of abstraction of grid resources. Each resource has its own capacity, for example, computing resource has two capacities, the power of CPU and the amount of a memory. There are two type of capacity. First, the capacity that will not be decreased when more user share the same capacity, e.g. information carried by some data. Second, the capacity that will be decreased when more user share the same capacity, e.g. memory. If total memory of a resource is $512\ MB$, and a request is granted for $128\ MB$ then the total amount capacity available or free is decreased to $384\ MB$ from $512\ MB$, and so on. The available computing power of a CPU can be identified by the percentage of unallocated computing capacity or by metrics such as MFlops or Tflops.

Peng Xiao et al. [54] added capacity issue in their relaxed model. They can limit the price of reservation violation as an impact of the relaxed reservation by adjusting parameter $v*$.

Sabitha et al. [55] also believe that estimating duration of execution time is very difficult if not impossible. As a result, applications tend to overestimate the duration of execution time to ensure their successful execution [33]. This behavior results in a high request rejection rate and a low resource utilization rate.

If any new request is coming for reservation, the rigid and elastic reservation will reject the request if there is no available compute node in the requested time-slot. But in relaxed resource advance reservation policy (RARP), overlapping of resources will take place and it will utilize even the small slots in between the resources [55]. A relaxed resource advance reservation policy (RARP) allows reservations to overlap each other under certain condition. They did not consider the reservation violation. They assume that the reservation violation will not occur because of the over estimating of the duration of execution time by including trust factor in their system.

They had a result from the experiment that by implementing relaxed resource advance policy (RARP), resource utilization is increased.

2.1.4 Flexible Advance Reservation (Static)

In flexible advance reservation, user jobs are scheduled within given flexible constraints. Start time is not fixed and could be varied in a time interval. Moaddeli et al. [44] has examined the impact of backfilling algorithm in a flexible advance reservation. Backfilling is proposed to improve the system utilization [56] [57]. The idea is to identify the idle compute nodes, i.e. times in which no jobs are assigned to one or more compute nodes. Sometimes, the job in the head of the waiting queue is bigger than the idle compute nodes, so it has to wait until there are idle compute nodes that fit this job. While waiting, backfilling allows the jobs in the waiting queue that is smaller than the job in the head of the waiting queue that fit in the idle compute nodes to move forward and executed on those idle compute nodes.

Backfilling policy including two major types, conservative and aggressive, is developed to enhance low utilization value of using FCFS as a whole scheduling policy, without losing fairness characteristic [38]. In aggressive or EASY (Extensible Argonne Scheduling System) backfilling, any job in the wait queue is permitted to be executed in those idle compute nodes as long as they are not delaying the waiting job at the head of the queue. It means that only the job at the head of the waiting queue has a reservation. On the other hand, in conservative backfilling, jobs are executed out of First Come First Serve (FCFS) order with the constraint of not delaying any waiting jobs. It means that all jobs in the waiting queue have a reservation but with a constraint that they can be moved forward to be executed before their reservation time. In their examination [44] they found that the more flexible the advance reservation the better resource utilization and the better response time of advance reservation. If the accuracy of duration of jobs is increased by twofold, then it decreases the resource utilization. Aggressive backfilling has better resource utilization than conservative backfilling.

Sulistio et al. [39] have done simulations of advance reservations in grid environments. Their results show that advance reservations interfere with on-demand requests by increasing their mean waiting time and by decreasing their resource utilization. This happens because advance reservations steal on-demand resources. To increase resource utilization and to minimize the rejection rate, Kaushik et al. [58] proposed a flexible reservation window scheme. Window size is the difference between the earliest start time of the user request and latest possible start time of the request. A window of size zero specifies a constrained request which can be satisfied only at the start time. By conducting extensive simulations, they conclude that, when the size of the reservation window is equal to the average waiting time in the on-demand queue, the reservation rejection rate can be minimized close to zero and increase resource utilization.

Real-time scheduling algorithms can be classified into online and offline algorithms based on whether they know or not about jobs that they have to schedule. In online scheduling algorithms, the scheduler does not know how many jobs should be scheduled and the scheduler does not know either regarding their constrains such as deadlines and computation times. Online scheduling algorithms make their scheduling decisions at runtime. In offline scheduling, on the other hand, the scheduler knows about how many jobs (and its constrains) will be scheduled. Scheduling decisions are based on fixed parameters assigned to jobs before their activation.

Castillo et al. [59] has proposed an online scheduling algorithm to increase system utilization using a concept from computational geometry. The computational geometry [60] is used to represent the time intervals corresponding to idle periods of each compute nodes i.e. the idle times between reserved for jobs whose requests were submitted earlier. Here deadline is equal or larger than the execution time. If the deadline is the same as the length of the job, it is called as immediate deadline, and if the deadline is longer than the execution time, it is known as general job deadline. Job scheduled in an idle period will create at most two new idle periods: one between the start of the original idle period and the start of the job (it is called the leading idle period), and one between the end of the job and the end of the original idle period (it is

called the trailing idle period). To schedule the general job deadline, they use the following strategies: Min-LIP, which minimizes the leading idle period; Min-TIP, which minimizes the trailing idle period; First-fit, which returns the first (i.e., earliest) feasible idle period, regardless of the sizes of the leading and trailing idle periods; LACT (latest available completion time), where the scheduler assigns an arriving job to the server with the latest completion time that is earlier than the ready time of the new job. LACT does not consider the idle periods created at each server between the times reserved for jobs whose requests were submitted earlier. The result shows that Min-LIP gives the best system utilization compared to other strategies.

2.1.5 Flexible Advance Reservation (Dynamic, Physical View)

This strategy takes an advantage of shifting even earlier reservations made (subject to given flexible constraints) to make room for new incoming reservation. Chunming et al. [53] introduced time span when there is a time span/range for the starting time of the job. This time span is called slack-time. They proposed a novel mechanism called FIRST (FlexIble Reservation using Slack Time). The advantage of slack-time is that the starting time of the job can be shifted to improve resource utilization and to reduce rejection rate. If new reservation comes the reservation system reschedules all unexecuted reservation one by one, according to the FIFO rule (First In First Out), min slack, min-min, min-max, and suffrage policy to see whether there is a solution or not. If there is no solution then the new reservation request will be rejected.

FIFO is the simplest selection strategy, where the request with the earliest will be selected to be scheduled. There is no other property considered. Min Slack is the strategy to select the next request to be scheduled which has the minimum slack time. Min-min based is the strategy that tries to find out the next request to be scheduled using the min-min algorithm [61], i.e. only the request with the minimum earliest finished time is selected to be scheduled. Min-max based is the strategy that tries to find out the next request to be scheduled using the min-max algorithm [61], i.e. only the request with the minimum latest finished time will be selected to be scheduled. The basic idea of Suffrage based strategy is to find out the next request to be scheduled

using the suffrage algorithm [62]. First, select an arbitrary request ra and ra is scheduled with the best allocation solution and record the earliest finished time as ta. Then try to schedule another request rb, and try to find out the best solution for ra again based on it and calculate the earliest finished time as tba (Generally $tba \geq ta$). Define the suffrage value of this request by $Suffrage = tba - ta$. Only the request with the maximum suffrage value will be selected as the next request to be scheduled.

They have implemented those algorithms through a simulation. Simulation results show that the better resource utilisation can be achieved by using slack time. Simulation results also show that min-min based request selection strategy has the best performance compare with other strategies such as the min-max, min slack time, FIFO and suffrage based strategy.

In FIRST as mentioned before, every time a new task comes, it reschedules the entire task. According to Netto et al. [63] rescheduling the task will have a better system utilization if the user can wait until 75% of waiting time has been passed, compared to 50% and 25%. They observe that the longer a user waits to fix their jobs the better the system utilization. In order to reschedule the task if the task t comes, firstly they separate tasks currently allocated into two queues: running queue and waiting queue. The first queue contains jobs already in execution and cannot be rescheduled. The second queue contains jobs that can be rescheduled. They try to reschedule the jobs in the waiting queue by sorting them first and then attempting to create a new schedule. They use five different sorting techniques: Shuffle, First in First out (FIFO), Biggest Job First (BJF), Least Flexible First (LFF) [64], and Earliest Deadline First (EDF). From the simulation they concluded that EDF technique has the best result in increasing resource utilization.

Behnam et al. [65] have introduced scheduling algorithm for advance reservation called GELSAR (Gravitational Emulation Local Search Advance Reservation) in grid system where the reservations can have a deadline (dj) which can be equal or larger than a ready time (rj) + length of the reservation (lj). GELSAR is a scheduling advance reservation using GELS method [66]. GELS takes the idea from

the natural principles of gravitational attraction. Gravity works in nature to cause object to be pulled toward each other. The more massive the object the more gravitational force it gets. The closer the object the more gravitational force it gets. Imagine that the search space is the universe and objects in this universe are the possible solutions. Each of these solutions has a mass represented by its objective function value. The higher the objective solution value the higher its mass. Every time the new reservation comes the GELSAR reschedule all reservation to find the best solution. If there is no solution the new incoming reservation is rejected. They compared their algorithm with other rescheduling algorithm (Genetic Algorithm, GA) and they found the result that their algorithm outperformed GA algorithm.

2.2 Data Structure

In administering advance reservations (admission control of advance reservations) in a Grid system, an efficient data structure is playing the important role in order to minimize the time for searching available compute nodes, adding and deleting a reservations. A user who requests an advance reservation will get a fast response time, in order to give a result whether his/her advance reservation request is accepted or not. There are several data structures for administering advance reservation which generally get categorized into two types i.e. time slotted data structure and continuous data structure. In time slotted data structure each request is stored in a certain number of consecutive fixed time intervals, also called time slots, where in continuous data structure each request defines its own time scale i.e. each advance reservation can start and finish at arbitrary times. Example of continuous data structure is Linked List and examples of time slotted data structure are segment tree and calendar queue data structure. Time slotted data structure approach has the advantage of restricting the amount of data that must be stored, i.e., the memory consumption is bounded and, furthermore, it can be easily implemented [67]. The majority of current implementations in the field of advance reservations support time slotted data structure [47] [68] [69] [70] [71] [72] [73].

2.2.1 Segment Tree Data Structure

A tree-based data structure is commonly used for admission control in network bandwidth reservation [71] [74] [75]. Each tree node contains a time interval and the amount of reserved bandwidth in its subtree. Therefore, a leaf node has the smallest time interval compared to its ancestor nodes. Hence, the amount of bandwidth required for a single reservation is stored into one or more nodes. In general, a tree-based structure has a time complexity of $O(log\, n)$ for searching the available bandwidth, where n is the number of tree nodes.

Brodnik et al. [71] have proposed Segment tree data structure for a network bandwidth reservation called Advanced Segment Tree (AST). AST is a modified segment tree [76]. Every node in the segment tree consists of the interval it represents ($IN = [SN, EN]$), pointers to each of the node's children, and two values i.e. nv and mv. nvN stores the amount of bandwidth that was reserved exactly the whole interval IN. mvN stores the maximum value of reserved bandwidth excluding the value nvN on interval IN.

The interval to which the root corresponds is M. L denotes the number of levels that the data structure consists of. All nodes on level l have time intervals of the same size and they do not intersect. They follow consecutively one another. This means that the union of all intervals on level l is M. N denotes the current node, NL denotes the left most child of N, and NR denotes the right most child. Let us see the example of bandwidth utility graph in Figure 2.1.

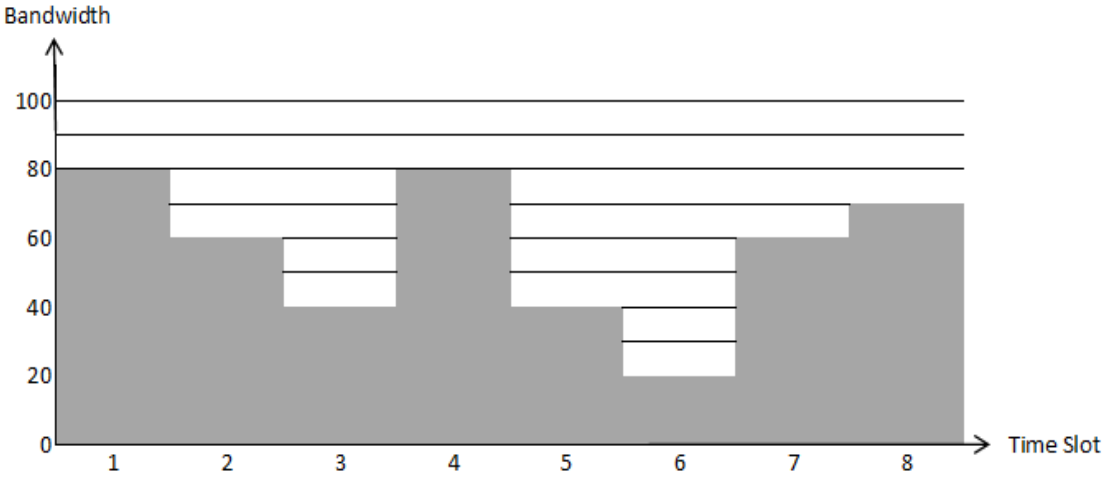

Figure 2.1: Example of Bandwidth Utility Graph

From Figure 2.1 we can see that in time slot number one reserved bandwidth is 80, time slot number two reserved bandwidth is 60, time slot number three bandwidth reserved is 40 and so on. Bandwidth capacity of Figure 2.1 is 100. Advance Segment Tree data structure for Figure 2.1 is shown in Figure 2.2.

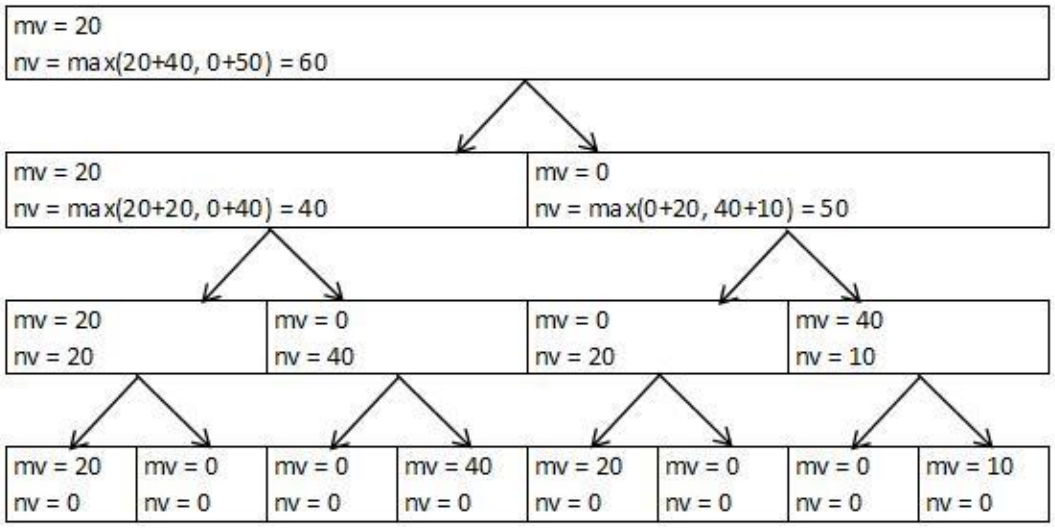

Figure 2.2: Example of Advance Segment Tree data structure

From Figure 2.2 we can see the value of mv and nv. The value of mv is determined from root node down to leaf nodes. Then the value of nv is determined form leaf node to root node. The value of nv for every leaf node is zero. Then the value of nv on the parent of leaf node (example node A) is maximum value $mv + nv$

between the children of node A (in this case leaf node). If a new reservation comes with bandwidth needed is 20 then the data structure of Advance Segment Tree can bee seen in Figure 2.3.

From Figure 2.3 we can see that a new reservation comes and needs a bandwidth 20 ($BR = 20$) and has an interval IR. We then start form the root node (Node A), since this reservation doesn't reserve all bandwidth range (time interval IA) denotes by root node ($IR < IA$) then we move down to its left child node. This left child node (node B, grey colour) has an $nv + mv = 80$. Free bandwidth in this node is $100 - 80 = 20$ ($BF = 20$). So we can allocate this new reservation in B node, because first; the new reservation can occupy whole interval IB ($IR \geq IB$), and second; B node has free bandwidth more or equal to what the new reservation needed ($BF \geq BR$). Update mv of node B become $mv = 40$.

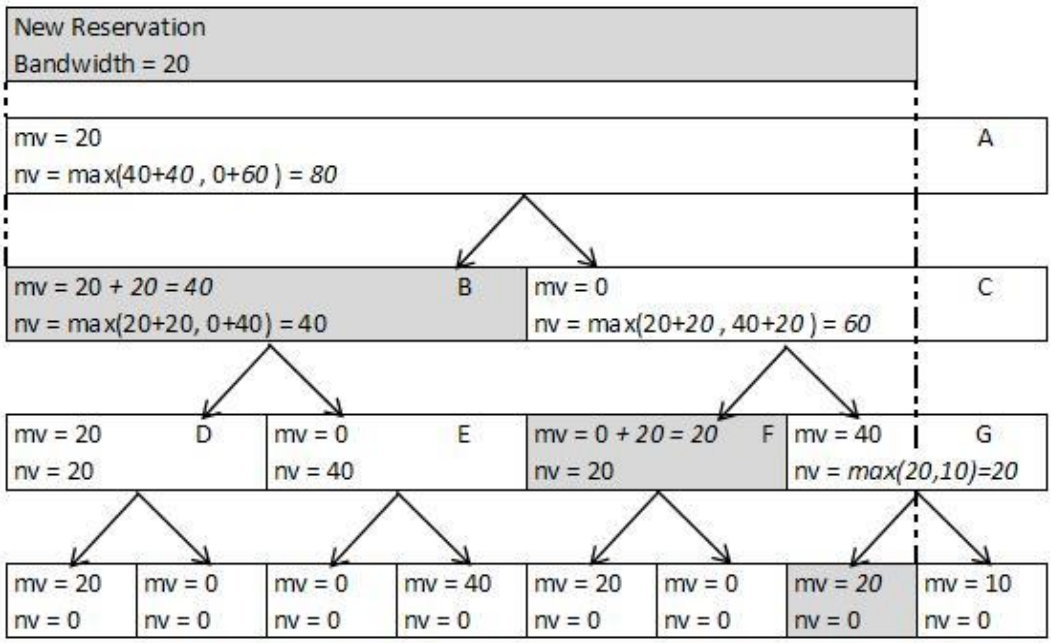

Figure 2.3: Scheduling a new reservation on advance segment tree data structure

The interval of new reservation is longer than an interval of B node, so the rest of the reservation that is not allocated yet (BL) is fall into the next node, Node C. Repeat the steps as if $IR = IR - IB$, and $BR = 20$ on node C instead of node A until $BL = 0$. We then have to update nv the data structure by updating all nv of

parent grey nodes up to the root node. The result is shown in Figure 2.3 where the italic font indicates the change of the value of the data structure.

2.2.2 Calendar Queue Data Structure

Brown et al. [72] have proposed Calendar Queue (CalendarQ), as a priority queue for future event set problems in discrete event simulation. It is modeled after a desk calendar, where each day or page contains sorted events to be scheduled on that period of time. A person schedules an event on a desk calendar by simply writing a future event on the appropriate page, one page for each day. There may be several future events scheduled on a particular day, or there may be no future event scheduled on a particular day. Priority of the scheduled event is the time at which an event is scheduled. The enqueue operation corresponds to scheduling an event. The earliest event on the calendar is dequeued by scanning the page for today's date and removing the earliest event written on that page.

Events can be scheduled for up to one year from the present date by making the calendar circular (circular array in data structure). If today is May 7, then we can schedule an event for January 3 of next year by simply flipping back to January 3 on the present calendar and writing the event on that page. If events are erased from the calendar as they are dequeued, it will be clear that the event on January 3 is scheduled for next year. When dequeueing the last event from December 31 of this year, we moves back to January 1 and begins dequeueing events scheduled for next year. We can use the same calendar many times. Notice that there is no need to move pointers or perform any other maintenance on the data structure as we moves from one year to the next year.

If we write the date beside each event, then events can be scheduled more than a year in advance. Suppose today date is May 7, 2013. If today's calendar page contains the entry "Register for IEEE conferences in Mumbai-June 7, 2014," it is clear that this event was scheduled more than a year in advance and that it should not be dequeued until next year. We have to check the date beside each event before dequeueing an event from today's calendar page, to assure that it is scheduled for this

year. If it is scheduled for this year, we can dequeue that event and erase it; if not, the event is ignored and left for next year or some year in the future.

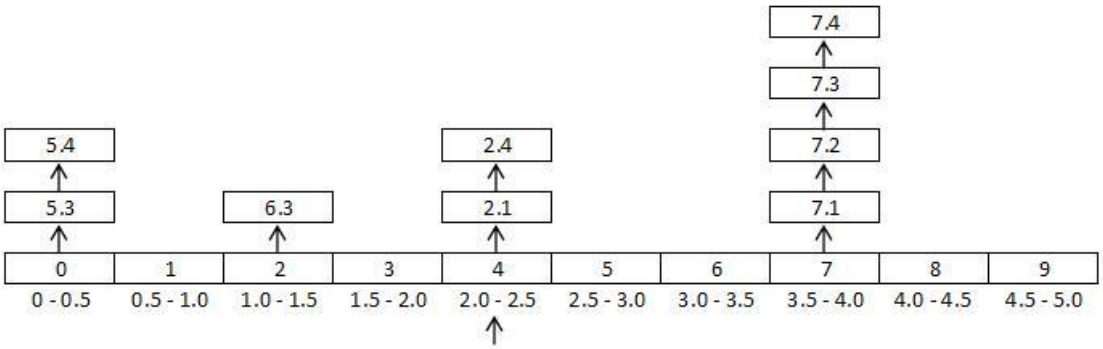

Figure 2.4: Ten day calendar Queue

Data structure to implement this CalendarQ can be represented by a sorted linked list of the event scheduled of that day. An array containing one pointer for each day of the year is used to find the linked list for a particular day. If the array name is "bucket," then $bucket[3]$ is a pointer to the list of events scheduled for the 3rd day of the year. The complete calendar thus consists of an array of 365 pointers and a collection of up to 365 linked lists.

Figure 2.4 shows an example of calendarQ with length of the year is 5 time units and the length of the day is 0.5 time units. The current year begins at 0.0 and ends at 5.0. Bucket 4 is the current date. Note that events scheduled in buckets 0 through 3 are scheduled for next year and are in the range $5.0 - 10.0$. The numbers on the below of the bucket are the time ranges for each day in the current year.

Calendar Queue has a complexity of $O(n)$ for adding reservations, where n is the number of reservations in the list for each bucket. Deleting reservations require a fast $O(1)$ because they are sorted in the list, and Calendar Queue only removes the reservations in the current bucket as time progresses.

2.2.3 Linked List Data Structure

Qing Xiong et al. [77] have proposed a linked-list data structure for advance reservation admission control. The idea for the data structure was partly illuminated by the bandwidth tree. Each bandwidth advance reservation request can be described with a series of parameters, e.g. start time, end time, bandwidth, duration, etc. They define the advance reservation request as $R = (bw,\ ts,\ te)$, where bw denotes the reserved bandwidth, ts and te denote the start and end time respectively. An example for reserved bandwidth is given in Figure 2.5.

The maximum available bandwidth in Figure 2.5 is 100 Kbps. The grey area area denotes the reserved bandwidth. A node of the linked list is defined as node $(bw,\ ts,\ pNext)$, where ts denotes the starting time of the time interval, $pNext$ denotes the pointer to the next node, bw denotes the reserved bandwidth during the time interval of current node. Thus the time interval covered by a node (denoted as $n1$) is from $n1.ts$ to $n2.ts$, where $n2$ is the next node to $n1$. The $pNext$ of the last node in a linked list is null. It means that the time interval covered by the last node ne is from $ne.ts$ to infinity. Usually the bandwidth of the last node is zero, because during the time interval covered by ne (from $ne.ts$ to infinity) no bandwidth is reserved. Figure 2.6 shows the data structure of a linked list recording the reservation information shown in Figure 2.5.

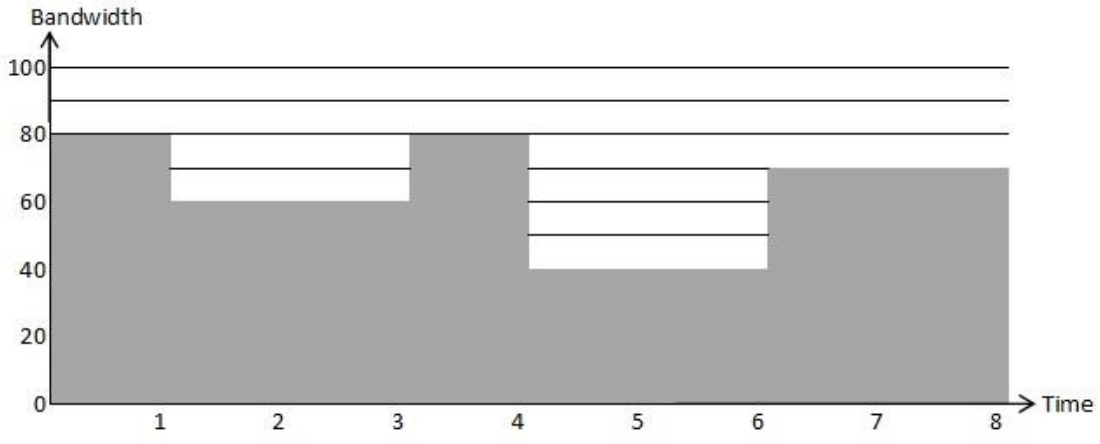

Figure 2.5: An example for the reserved bandwidth. The time ranges from 0 to 8

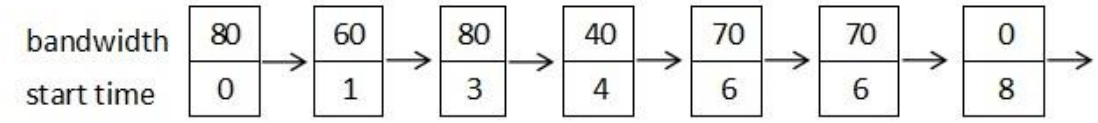

Figure 2.6: Data structure of a linked list corresponding to the reservation example shown in Figure 2.5 Each node contains the starting time, reserved bandwidth and a pointer to the next node

The process of performing admission control with linked list is described as follows. Initially there is only a single node i.e. head node written as $head(0, 0, null)$ in the linked list. When a reservation request $R(bw, ts, te)$ arrived and the request is admitted (when $R.bw \leq BWmax$), 1 new node will be added into the linked list if $R.ts$ is equal to 0. If $R.ts$ is greater than zero than 2 new nodes will be added into the linked list. Positions of new nodes in the linked list are determined by their starting time. With this situation every time a new reservation request comes we have to do checking and adding operations. To avoid searching the proper position from the head node of the linked list time and time again, they utilize a pointer as a local head node to mark the start node that has been searched from for the prior request. In order to implement it, they should collect all the reservation requests during a given period (this is called offline scheduling), sort them by their starting time and then perform the admission control process for them together. Utilizing this mechanism will decrease the time for searching the proper starting node greatly.

Linked List data structure has a sequential searching method leading to $O(totAR)$ time complexity, where $totAR$ is the total number of reservations. This is because the List does not partition each reservation into a fixed time interval like a tree-based structure. Linked List can become very inefficient for running these operations on many short reservations, because it needs to find the correct position or node starting from the root node.

Qing Xiong et al. [77] compared the performance of their data structure (linked list) for bandwidth advance reservation admission control with bandwidth tree data structure proposed by Tao Wang et al. [78] and time slotted array proposed by

Burchard et al. [68]. They had a result that linked data structure has a better performance for admission control in bandwidth advance reservation.

2.2.4 GarQ Data Structure

Sulistio et al. [73] have proposed GarQ (Grid Advance Reservation Queue) for administering advance reservation in grid system. GarQ is a modified Calendar Queue (CalendarQ) data structure. GarQ has buckets with a fixed time interval denotes by δ, which represents the smallest slot duration, as with the Calendar Queue. This bucket is an array of a node containing rv (the number of already reserved CNs in this bucket) and a linked list (sorted or unsorted) containing the reservations which start in this bucket node.

Figure 2.7 shows the example of reserved compute node on the grid system with 5 compute nodes represented in a time-space diagram and the size of time slot is 1 time unit ($\delta = 1$). User1 scheduled at compute node $cn0$ from time 5 and up tome time 8. User 2 scheduled at compute nodes cn1 and compute node $cn2$ from time 6 up to time 7. User 3 scheduled at compute nodes $cn3$ and compute node $cn4$ from time 5 up to time 8. User 4 scheduled at compute nodes $cn1$ from time 7 up to time 10. User 5 scheduled at compute nodes $cn1$ and compute node $cn2$ from time 10 up to time 12.

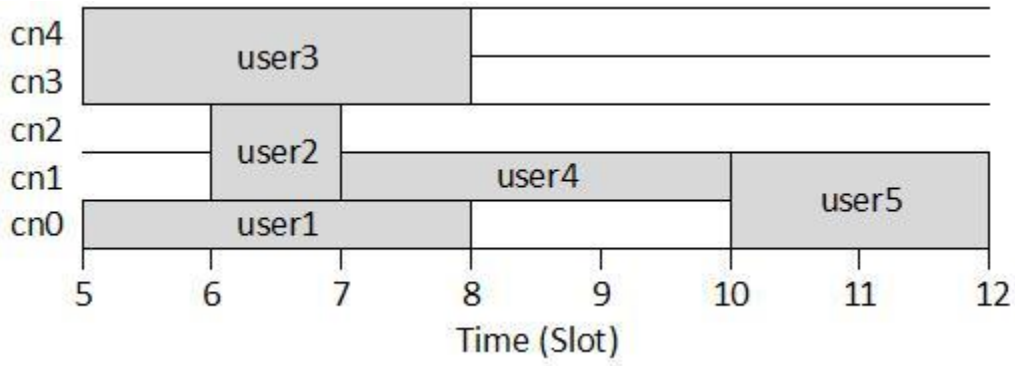

Figure 2.7: Example of reserved compute node in time space diagram with 5 compute nodes

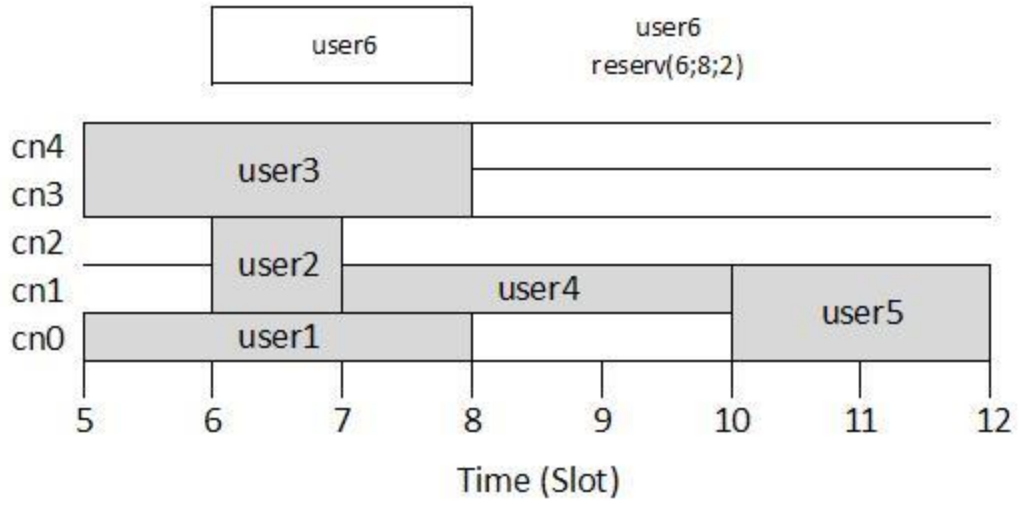

Figure 2.8: User 6 arrives, request 2 compute nodes from time 6 up to time 8, defined by $reserv(6;\ 8;\ 2)$

When a new request from User 6 arrives, the system checks for any available compute nodes. In this example, the request is defined as $reserv(ts;\ te;\ numCN)$. User 6 has a request as $reserv(6;\ 8;\ 2)$. However, only one node is available at interval 7 and 8, hence, this request will be rejected. Figure 8 shows user6 request in time space diagram of Figure 2.7.

Figure 2.9 shows how reservations are stored in GarQ with Sorted Queue with δ = 1 time interval, by using the example illustrated in Figure 2.7.

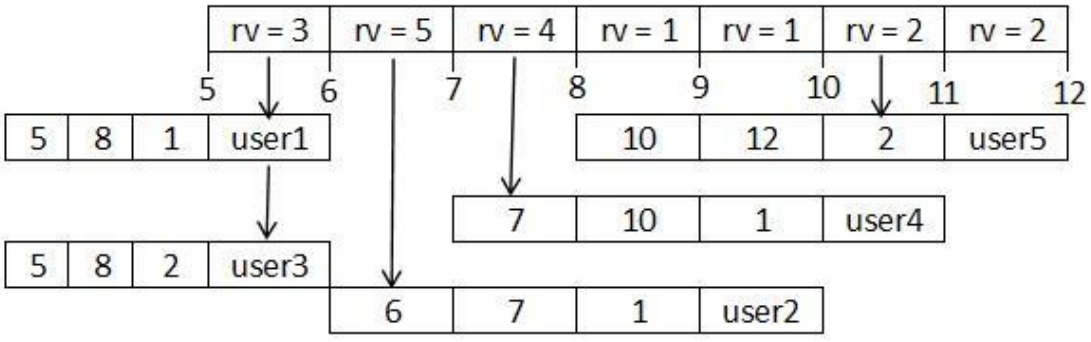

Figure 2.9: A representation of storing reservations in GarQ with Sorted Queue and δ = 1

The complexity of GarQ for searching and deleting reservation is $O(exec)$, where $exec$ is the number of buckets needed to execute the job. The complexity for adding reservation is $O(exec)$ and $O(k + exec)$ when using unsorted and sorted queue respectively, where k is the number of reservations in a bucket list. Sulistio et al. [73] compared the performance of GarQ, CalendarQ, Segment Tree, and Linked List data structure. They had a result that GarQ performed better among the other data structure.

2.3 Revenue Management System

In grid system, the resource owners (producer) and resource users (consumers) have different objectives, strategies, and supply-and-demand patterns. Resource owners and resource users are geographically distributed in different time zones. The most commonly used approaches for managing such complex environments are driven by system-centric and user-centric policies [42]. System centric is a traditional approach for a resource management that attempts to optimize system-wide measure of performance. Also system centric is usually used in managing resources in single administrative domains. On the other hand, User-centric approaches focus on delivering maximum utility of the resources to the users based on their QoS requirements, i.e., an assurance of performance based on the criteria that the user thinks important such as the deadline by which his jobs have to be completed. Implementing QoS requires a system of rewards and penalties and, hence, typically user-centric approaches are driven by economic principles.

System-centric Grid resource management systems such as Legion [79], Condor [80], AppLeS PST [81] [82], Net-Solve [83], PUNCH [84], and XtremWeb [85] adopt a conventional strategy, where which jobs are to be executed at which resources is scheduled based on cost functions driven by system-centric parameters. Their goal is to increase the system throughput, utilization, and complete execution as soon as possible, rather than increasing the utility of application processing. They treated that the resource access cost for executing the job to be the same. In reality the resource access cost is higher if there is a deadline to complete the job. On the other

hand, users do not want to pay the highest resource access cost, but users want to search or select a particular price based on the demand, priority, and available budget.

In an economy-based approach, the end-users requirements direct and drive the scheduling decision dynamically at runtime. On the other hand, without economy model the resources access cost is driven by software and hardware cost. Pricing policies are based on the supply of resources and the demand from the users is the main driver in the competitive, economic market model. Therefore, a resource owner competes with other resource owners and a user competes with other users.

Buyya et al. [42] have introduced a Grid economy concept. This concept provides a mechanism for regulating supply and demand, and calculates pricing policies based on these criteria. Grid economy concept offers an incentive for resource owners to join the Grid, and encourages users to utilize resources optimally and effectively. In general, the benefits of grid economy can be listed as follows:

- Grid economy offers uniform treatment of all resources. That is, it allows trading of everything including computational power, memory, storage, network bandwidth/latency [86], data, and devices or instruments
- Grid economy helps in regulating the supply and demand for resources
- Grid economy offers incentive for resource owner to contribute their resources for others to use and take a profit from it, so encourages a large scale grid.
- Grid economy encourages the solution of time critical problems first then offers an economic incentive for users to reduce their deadline or priority by paying less.
- Grid economy provides a media for a user to compare the resource access cost.
- Grid economy supports a simple and effective basis for offering differentiated services for different applications at different times.

- With grid economy decision-making process is distributed across all users and resource owners, so it helps in building a highly scalable grid.

Several studies [43] [44] [45] [53] [58] [59] [63] [65] [87] have been done to improve scheduling and handling of reservations in grid systems using different techniques. However, a simple pricing model to determine the usage cost of each reservation has been provided by [43] [44] [45]. To increase incentives or profits, resources provider might need to adopt a more complex pricing model. The goal of revenue management is to maximize profits by providing the right price for every product to different customers (it may have two different classes of customer which have different price, i.e. full-price and discount-price), and periodically update the prices in response to market demands [46]. In revenue management system, booking limit is the maximum number of computing resources that may be reserved at each price class. If we set the discount booking limit to low, we will reject discount customers but the full-price customers are not enough to reserve all resource capacity. This is called spoilage, since computing resource becomes spoiled by rejecting discount-price customers. On the other hand, if we allow too many discount-price customers to book, we have a risk of rejecting full-price customers which give us more profit. This scenario is called dilution, where we dilute the revenue we can have from saving an additional computing resources for a high-price customers. The challenge of capacity allocation is to balance the risks of spoilage and dilution to maximize the revenue [46].

2.4 Discussion and Summary

In this chapter we have discussed literature review regarding advance reservation in grid system. Many studies have been done to improve scheduling and handling of reservations in grid systems using different techniques. We have broadly categorized them into five categories. The first category is rigid reservation, where if a reservation request does not get available compute nodes to satisfy its requirement, then the system will reject it. The second category is elastic reservation, where if a reservation request does not get available compute nodes to satisfy its requirement, then the

system will search alternative available compute nodes and send it to user. Then user resubmits his/her advance reservation request with an alternative requirement given by resource provider only if user agrees with the alternative requirement. The third category is overlapping/relax advance reservation, to decrease the rejection rate or to increase the resource utilization. The fourth category is flexible advance reservation, where the advance reservation start time requirement is not strict. The advance reservation can start in certain interval time defined by a user. In this case, the resource provider tries to find available nodes in that interval time without changing the previous advance reservation already scheduled (static scheduling). The fifth category is flexible advance reservation dynamic scheduling, physical view, where if flexible advance reservation comes, the resource owner will reschedule the entire job that have been allocated but not executed yet.

An efficient data structure is playing the important role in order to minimize the time for searching available compute nodes, adding and deleting an advance reservations. In this chapter we have presented several type of data structure for administering advance reservation. These data structure are: Segment Tree, Calendar Queue (CalendarQ), Linked List and GarQ.

In this chapter we also presented literature on the need of revenue management to increase the revenue of resource owner by providing the right price for every product to different customers, and periodically updating the prices in response to market demands.

3 A Proposed Advanced Planning and Reservation Strategy

We propose a novel advance planning and reservation strategy namely First Come First Serve Ejecting Based Dynamic Scheduling (FCFS-EDS) to increase resources utilization in a grid system. To achieve this we introduce a new notion that maps a user job to a virtual compute nodes (called logical view) which are subsequently mapped to actual compute nodes (called physical view) at the time of execution. An FCFS-EDS's lemma ensures the success of such a mapping with increased throughput. This approach can be categorized under Flexible Advance Reservation (Dynamic, Logical View).

3.1 Flexible Advance Reservation: A new Perception

Parallel applications have very large resource requirements and require resources from multiple parallel computers to execute in the same time. Other applications are workflow applications, where the job in workflow application needs to wait of the other job before it can be executed. This is called dependency in workflow applications. To ensure a smooth broadcast of video and audio over the network, multimedia or soft real-time applications, such as video conferencing, need to have a certain amount of bandwidth which is not tolerable if there are any dropouts in a network transfer. With the three aforementioned applications, it is clear that they need

to have to an advance reservation to ensure that the specified resources are available for application when required [41]. In most grid systems with traditional scheduler, submitted jobs are placed in the wait queue if mandated resources are not available for them. Every grid system may use a different scheduling algorithm, for example First Come First Serve (FCFS), Shortest Job First (SJF), Earliest Deadline First (EDF), or EASY Backfilling [38] that executes jobs based on different parameters, such as number of resources, submission time, and duration of execution. With these scheduling algorithms, there is no guarantee about when these jobs will be executed [39]. Advance reservation is the process of requesting resources for use at specific times in the future [41]. Common resources that can be reserved or requested are compute nodes, network bandwidth, storage elements or a combination of any of those.

In flexible advance reservation, user jobs are scheduled within given flexible constraints. Start time is not fixed and could be varied in a time interval. Let us define a new perception of "flexible advance reservation" as follows: "Flexible advance reservations are reservations where the time between reservation request starting time (t_{es}) and reservation request end time (t_d) is longer than the execution time (t_e) of a job" as shown in Figure 3.1.

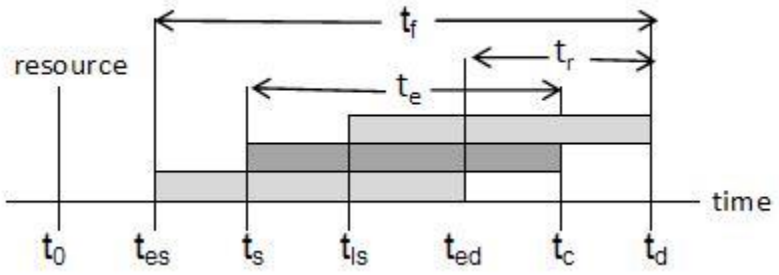

Figure 3.1: Flexible Advance Reservation

t_0 : current time

t_{es} : lower bound to start time of the job (earliest start time)

t_d : upper bound to end time to execute the job

t_e : execution/duration time of the job

t_r : relax time, define by $t_r = t_d - t_{es} - t_e$

t_{ls} : upper bound to start execution of the job (latest start time), defined as $t_{ls} = t_d - t_e = t_{es} + t_r$

t_{ed} : lower bound to end time to execution of the job, defined as $t_{ed} = t_{es} + t_e$

t_s : start time to execute the job ($t_{es} \leq t_s \leq t_{ls}$)

t_c : completion time to execute the job ($t_{ed} \leq t_c \leq t_d$)

t_f : flexibility time, defined as $t_f = t_d - t_{es}$

f : the degree of flexibility, define as $f = \frac{t_f}{t_e}$, here $f \geq 1$, (if $f = \infty$, the job considered as non reservation job mode. If $t_s = t_0$ and $f = 1$ the job for reservation considered with top most priority leading to immediate reservation mode i.e., scheduling mode)

$userId$: identification of a user

$jobId$: identification of a job

$numJ$: number of jobs

$numCN$: number of compute nodes needed

$maxCN$: total number of compute nodes

From the above aforementioned variable, we don't use some of them explicitly i.e. t_0, t_d, t_{ed}, t_f, and f but we define it to give a clear understanding regarding our new perception of advance reservation.

In chapter 2, we have attempted to broadly categorize these advance reservation strategies reported in the literature which are: Rigid reservation [47] [48], Elastic Advance Reservation [43], Overlapping/Relax Advance Reservation [50] [54] [55], Flexible Advance Reservation (Static) [44] [58] [59], Flexible Advance Reservation (Dynamic, Physical View) [53] [63] [65]. Our proposed advance

reservation scheduling strategy namely First Come First Serve Ejecting Based Dynamic Scheduling (FCFS-EDS) falls into Flexible Advance Reservation (Dynamic, Logical View). Table 3.1 presents the parameters that have been used by other researchers. The rows indicate possible parameter and column the scheduling strategies. The entries are notations/symbols, policy and name wherever provided by the authors otherwise dash has been used.

Table 3.1: Comparison between other scheduling strategies

	Our work	Sulistio et al.	Castillo. et al.	Kaushik et al.	Xiao et al.	Chunming et al.	Netto et al.	Behnam et al.	Moadeli et al.
Current Time	t_0	-	-	-	t_0	-	-	$t = 0$	-
Time of the earliest start time of the job	t_{es}	t_{is}	r_j	-	t_{start}	t_{is}	t_r	r_j	t_{rt}
start time to execute the job	t_s	-	-	t_{start}	-	t_{start}	t_s	-	t_{st}
completion time to execute the job	t_c	-	-	t_{end}	-	-	t_c	-	t_{ct}
the end time to execute the job	t_d	t_{ie}	d_j	-	-	-	d	d_j	t_{dt}
execution time of the job	t_e	dur	l_j	dur	d	$duration$	t_e	L_j	t_{ad}
Relaxed time	t_r	-	-	W	-	$slack$	-	-	-
Degree of flexibility	f	-	-	-	-	-	-	-	-
Compute nodes needed	n	n	1	1	1	1	N		N
Name	Flexible	elastic	general job deadline	unconstrained /flexible	relaxed	flexible	flexible	-	Flexible
Scheduling	FCFS - EDS	FCFS -OSP	FCFS Min-Lip	FCFS	FCFS	Min-min	EDF	GELSAR	FCFS

3.2 Types of Jobs

Our advance planning and scheduling strategy for advance reservation FCFS-EDS can handle two types of job. It can handle parametric jobs and MPI Jobs.

3.2.1 Parametric jobs

Parametric jobs are similar jobs that only differ in arguments or input/output files. It is good to use parametric job type, because submitting them separately can take much time. With parametric job type, we can submit bulk of jobs as a single job.

In science and engineering parametric computational experiments become very important as a means of exploring the behavior of complex systems. For example, the behavior of wing in airplane can be explored by running computational model of the airfoil multiple time, where, each time of running this computational model parameters such as angle of attack, air speed, etc. are varied [8].

3.2.2 MPI jobs

To solve compute intensive applications faster we can use three strategies which are : first, using faster hardware with high performance processor (work harder), second; using more efficient and faster algorithm (work smarter), and third; using multiple computer to solve a particular task (get help) [88]. Using faster hardware or high performance computing seem to be a good solution because of Moore's Law [89] which stated that the number of transistors on integrated circuits doubles approximately every two years. But still this high performance computer with efficient algorithm is not enough to speed up the completion of compute intensive application. One possible solution to speed up the completion of compute intensive application is to connect multiple processors together and coordinate their computational effort.

The main reason for creating and using parallel computers is that parallelism is one of the best ways to overcome the speed bottleneck of a single processor [90]. There are three strategies for creating parallel applications. The first strategy is based

on automatic parallelization. With this strategy a programmer does not need to bother about parallelizing task. The second is based on the use of parallel libraries. With this strategy parallel code that is in common to several applications is encapsulated into a parallel library. The third strategy is major recoding or resembles the code from-scratch in developing a parallel application. The programmer is free to choose the language and the programming model used for developing a parallel application [91].

Message Passing Interface (MPI) is an API specification that allows parallel applications to communicate with each other by sending and receiving messages. It is typically used for parallel programs running on computer clusters and supercomputers. Parallel applications then can be called as MPI application/job. If MPI job need n compute nodes to execute its job, this n job must start at the same time.

3.3 FCFS-EDS Strategy

This strategy takes an advantage of shifting earlier reservations made (subject to given flexible constraints) to make room for new incoming reservation request. However the planning for reservation in our proposed model provides a logical view as against the physical view reported in the literature. Therefore we call our method as First Come First Serve-Ejecting Based Dynamic Scheduling (FCFS-EDS) strategy.

3.3.1 Model

User provides the following parameter in his reservation:

a. $userId$: identification of a user
b. $jobId$: identification of a job
c. t_{es} : lower bound to start time of the job (the earliest start time)
d. t_{ls} : upper bound to start execution of the job (the last start time)
e. t_e : execution time of the job
f. $numCN$: number of compute nodes needed

Figure 3.2 shows Planning Module. In this module FCFS-EDS scheduling strategy communicates to data structure to see whether the incoming reservation can be allocated or not using first fit scheduling strategy. If there are compute nodes for this incoming reservation, FCFS-EDS scheduling module allocate this reservation in logical view and save it in data structure. If it cannot be allocated then FCFS-EDS scheduling strategy communicates to data structure to see whether there are allocated reservations in logical view that can be shifted to give a room to a new incoming reservation. If there are, shift them and allocate the new incoming reservation in logical view and save it in data structure.

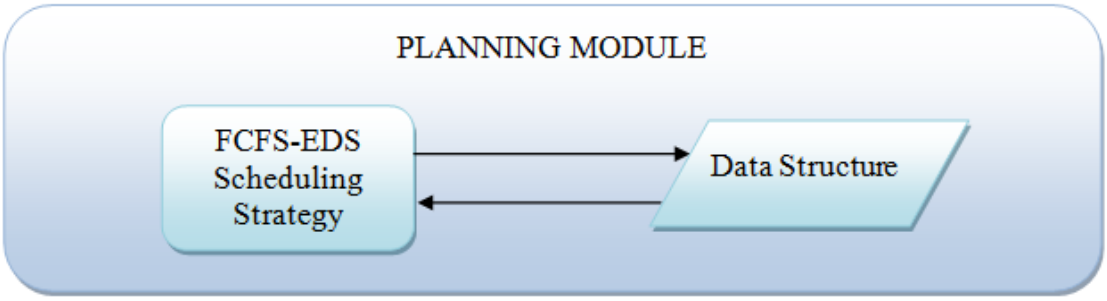

Figure 3.2: Planning Module

3.3.2 Algorithm

Algorithm of FCFS-EDS for parametric job is depicted in Algorithm 3.1 and explained as follows. Let us assume that earlier "$n-1$" reservation requests made to FCFS-EDS, specified by the following parameters: t_{es}, t_{ls}, t_e, $userId$, $jobId$, have been successfully scheduled. Definition of variables is explained in lines 4-9. Lines 10–14 initialize the variables.

Incoming "n^{th}" reservation request is scheduled based on first fit strategy (iteration of lines 16-25). It tries to search for resources within the given constraints (t_{es} and t_{ls}) and without disturbing plan for previous "$n-1$" reservation requests that were made. Let us call this as plan P_{old}. If within the given flexible constraints the search fails to allocate resources the algorithm tries to move around previous "$n-1$" reservations to accommodate "n^{th}" reservation request (lines 31-44). If the resources

are found then the search is declared successful and the algorithm outputs a new plan P_{new} that depicts "n" reservation requests as a logical view. If the search is a failure then the "$n-1$" reservation request plan has to be restored to its previous state i.e. P_{old}.

Time complexity for Algorithm 3.1 can be calculated with the assumption that all basic operations have $O(1)$, hence assignment operation has $O(1)$, comparison operation has $O(1)$, addition operation has $O(1)$ and so on, hence we have:

$Line\ 10-14 = O(1)+O(1)+O(1)+O(1)+O(1) = O(1)$

$Line\ 16 = O(1)$

$Line\ 18 = O(te)$

$Line\ 19 = O(1)$

$Line\ 20\ and\ 21 = O(te)+O(1) = O(te)$

$Line\ 23-25 = O(1)+O(1)+O(1) = O(1)$

$Hence\ for\ line\ 16-25 = (tr)\{O(1)+O(te)+O(1)+max(O(te),O(1))\}$

$$= (tr)\{O(1)+O(te)+O(te)\}$$

$$= (tr)\{O(te)\}$$

$$= O(te.tr)$$

$Line\ 28-30 = O(1)+O(1)+O(1) = O(1)$

$Line\ 31 = O(1)$

$Line\ 33 = O(te)$

$Line\ 34 = O(1)$

$Line\ 37\ and\ 38 = O(te)+O(1) = O(te)$

$Line\ 41-44 = O(1)+O(1)+O(1)+O(1) = O(1)$

$$Hence\ Line\ 28-44 = O(1) + (tr)\{O(1) + O(te) + O(1) + max(O(te), O(1))\}$$

$$= (tr)\{O(te) + O(te))$$

$$= (tr)(O(te))$$

$$= O(tr.te)$$

$$Line\ 47 = O(tr)$$

Hence Algorithm, 3.1, FCFS-EDS has time complexity

$$= O(1) + O(te.tr) + O(te.tr) + O(tr)$$

$$= 2.O(te.tr) + O(tr)$$

$$= O(te.tr) + O(tr)$$

$$= O(te + 1).O(tr)$$

$$= O(te).O(tr)$$

$$= O(te.tr)$$

Algorithm 3.1: FCFS-EDS for papametric job

```
1 Function searchAndAlloc(userId, jobId, tes, tls, te : integer) → boolean
2 //search and allocate job with given tes, tls, te
3  Dictionary :
4  ts : integer /*start time of the job*/
5  tc : integer /*finish time of the job*/
6  min : integer  /*minimum free nodes within interval  ts - tc*/
7  t   : integer  /*timeslot of minimum available node between ts to tc*/
8  tr  : integer  /*relax time, length between of tes and tls*/
9  relax : integer /*different between ts and tes time (ts - tes + 1)*/
Algorithm :
10 tr ← tls – tes
11 succeed ← false
12 ts ← tes
13 tc ← tes + te – 1
14 relax ←  ts - tes
15
16 while (!succeed and (relax≤tr)) do  /*searching by first fit strategy*/
17    /*searching minimum free node between ts to tc*/
18    min,t ← minFreeNode(ts, tc)
19      if(min > 0) then
20        add(userId, jobId, tes, ts, tls, te)
21        succeed ← true
```

```
22      else
23        ts ← t + 1
24        tc ← ts + te - 1
25        relax ← ts - tes
26 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
27
28 ts ← tes
29 tc ← tes + te - 1
30 relax ←  ts - tes
31 while (!succeed and (relax≤tr)) do
32    /*searching minimum free node between ts to tc*/
33    min,t ← minFreeNode(ts, tc);
34    if(min > 0) then
35      /*push or plan the job to data structure and update free node
36      between ts to tc*/
37      add(userId, jobId, tes, ts, tls, te)
38      succeed ← true
39    else
40      /*try to shift a job that start at t time slot*/
41      if(!shiftNode(t)) then //can't be shifted, move ts to t+1
42        ts ← t + 1
43        tc ← ts + te - 1
44        relax ← ts - tes
45 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
46 if (!succeed)
47   putBackAllShiftedJob()
48 return succeed
```

Algorithm of FCFS-EDS for MPI job is depicted in Algorithm 3.2 and explained as follows. Let us assume that earlier "$n-1$" reservation requests made to FCFS-EDS, specified by the following parameters: t_{es}, t_{ls}, t_e, $userId$, $jobId$, $numCN$ have been successfully scheduled. Definition of variables is explained in lines 4-9. Lines 10–14 initialize the variables.

Incoming "n^{th}" reservation request is scheduled based on first fit strategy (iteration of lines 16-25). It tries to search for resources within the given constraints (t_{es} , t_{ls} and $numCN$) and without disturbing plan for previous "$n-1$" reservation requests that were made. Let us call this as plan P_{old}. If within the given flexible constraints the search fails to allocate resources the algorithm tries to move around previous "$n-1$" reservations to accommodate "n^{th}" reservation request (lines 31-44). If the resources are found then the search is declared successful and the algorithm outputs a new plan P_{new} that depicts "n" reservation requests as a logical view. If the search is a failure then the "$n-1$" reservation request plan has to be restored to its previous state i.e. P_{old}.

The main difference with respect to the parametric job is that this MPI job must start its jobs in the same time slot. Instead of checking available one compute node it checks $numCN$ available compute nodes to satisfy the MPI reservation request. $numCN$ is the number of compute nodes requested by a user.

Algorithm 3.2: FCFS-EDS for MPI job

```
1 Function searchAndAlloc(userId, jobId, tes, tls, te, numCN : integer) →
boolean
2 //search and allocate job with given tes, tls, te, numCN
3  Dictionary :
4  ts : integer /*start time of the job*/
5  tc : integer /*finish time of the job*/
6  min : integer  /*minimum free nodes within interval  ts - tc*/
7  t   : integer  /*timeslot of minimum available node between ts to tc*/
8  tr  : integer  /*relax time, length between of tes and tls*/
9  relax : integer /*different between ts and tes time (ts - tes + 1)*/
Algorithm :
10 tr ← tls – tes
11 succeed ← false
12 ts ← tes
13 tc ← tes + te – 1
14 relax ←  ts - tes
15
16 while (!succeed and (relax≤tr)) do  /*searching by first fit strategy*/
17    /*searching minimum free node between ts to tc*/
18    min,t ← minFreeNode(ts, tc)
19      if(min ≥ numCN) then
20        add(userId, jobId, tes, ts, tls, te, numCN)
21        succeed ← true
22      else
23        ts ← t + 1
24        tc ← ts + te - 1
25        relax ← ts - tes
26 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
27
28 ts ← tes
29 tc ← tes + te – 1
30 relax ←  ts - tes
31 while (!succeed and (relax≤tr)) do
32    /*searching minimum free node between ts to tc*/
33    min,t ← minFreeNode(ts, tc);
34    if(min ≥ numCN) then
35      /*push or plan the job to data structure and update free node
36      between ts to tc*/
37      add(userId, jobId, tes, ts, tls, te, numCN)
38      succeed ← true
39    else
40      /*try to shift a job that start at t time slot*/
41      if(!shiftNode(t, numCN-min)) then //can't be shifted, ts equal t+1
42        ts ← t + 1
43        tc ← ts + te - 1
44        relax ← ts - tes
45 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
46 if (!succeed)
47   putBackAllShiftedJob()
48 return succeed
```

3.3.3 Planning (Logical View) to Reservation (Physical View) Mapping

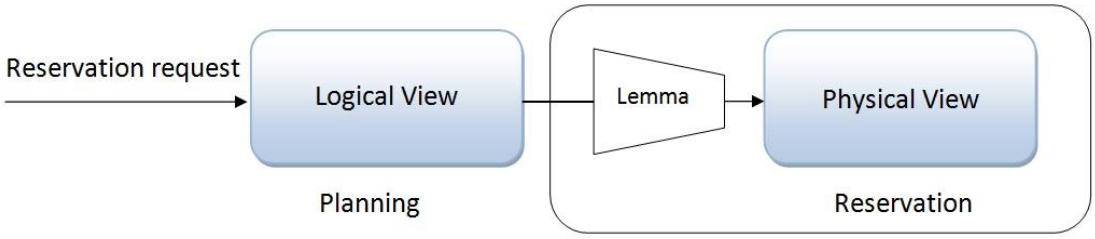

Figure 3.3: Planning (Logical View) to Reservation (Physical View) Mapping

As shown in Figure 3.3, Reservation Module maps allocated reservation in logical view to a physical view. This mapping has to be done in a way that all reservation allocated in logical view (a plan) will get an actual compute node, and the reservation that has started at a particular compute node has to be executed on that particular node for the entire time slot. In order to achieve this, we introduce an FCFS-EDS's lemma to guarantee that the plan (logical view) can always be mapped into actual compute node (physical view), and once a job is started at certain compute node, then it will be executed on the same dedicated compute node for the entire time slot. Applying the FCFS-EDS's lemma (discussed later) to the plan as depicted in Figure. 3.4, we guarantee that all the jobs will be executed as shown in Figure 3.5 (Physical View).

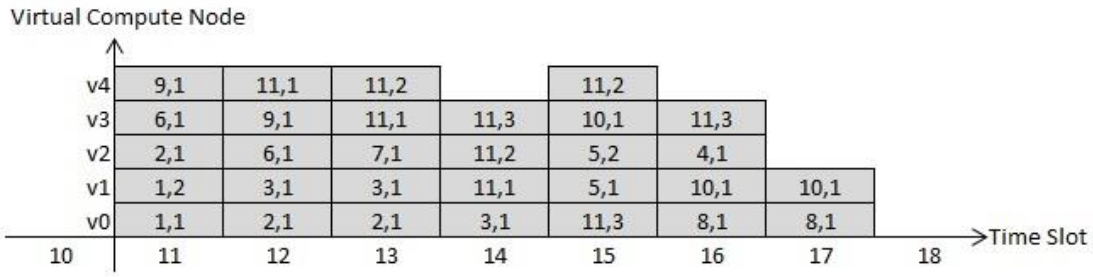

Figure 3.4: Eleven users have been allocated using FCFS–EDS (Logical View)

Figure 3.4 shows allocated reservation in logical view (plan), where x axis indicates time slot, and y axis indicates virtual compute node (Logical View). As there are 5 virtual compute nodes, they are shown along the y axis as $v0, v1, v2, v3$, and $v4$. Reservations of eleven users have been allocated during time slots 11 to 17. From

Figure 3.4 we can see that the virtual compute nodes allotted to $userId = 9$ and $jobId = 1$ are $v4$ at time slot 11 ($t_{es} = 11$) and $v3$ at time slot 12 as only 1 job (requiring 2 execution time slots) has been submitted by this user ($userId = 9$).

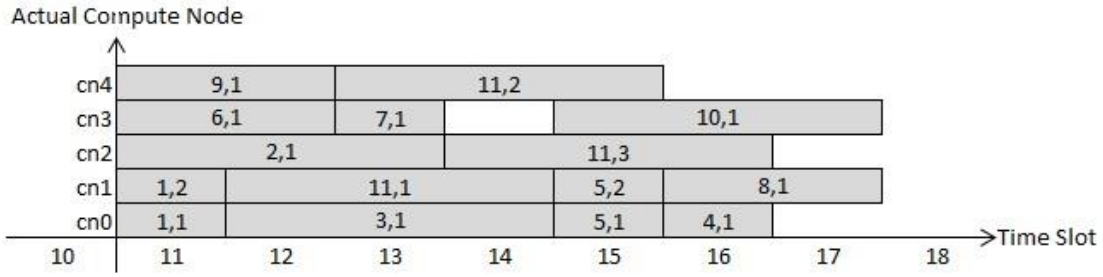

Figure 3.5: Actual Compute Node Mapping (physical view) at $t_0 = 18$

Figure 3.5 shows allocated reservation in physical view, where x axis indicates time slot, and y axis indicates actual compute nodes (Physical View). As there are 5 actual compute nodes, they are shown along the y axis as $cn0, cn1, cn2, cn3$, and $cn4$. From Figure 3.5 we can see that the actual compute node allotted to $userId = 9$ and $jobId = 1$ is $cn4$ at time slot 11 ($t_{es} = 11$) and at time slot 12 as only 1 job (requiring 2 execution time slots) has been submitted by this user ($userId = 9$).

From Figure 3.4 we can see that allotted compute nodes for $userId = 9$ and $jobId = 1$ are $v4$ at time slot 11($t_{es} = 11$) and $v3$ at time slot 12. It is seen that the same job on different time slot is allocated on different virtual compute node. Upon applying the FCFS-EDS's lemma, the actual compute node allotted to $userId = 9$ and $jobId = 1$ is $cn4$ at time slot 11($t_{es} = 11$) and at time slot 12. It is seen that once a job is started at certain compute node, then it will be executed on the same compute node for the entire time slot.

The concept of proof of our advance planning and scheduling (reservation) strategy is assured due to the following FCFS-EDS's Lemma:

FCFS-EDS's Lemma: *If there is a plan for scheduling a job on the consecutive time slot on virtual compute nodes (which are selected freely) (Logical View) then it guarantees that the job will be executed on a dedicated physical node for the required execution time (Physical View).*

Proof:

Let $J(t)$ be is an array of size $maxCN$ (maximum number of compute node) scheduling plan at time slot t. $J(t)(i)$ is i^{th} element of $J(t)$, and it holds the job id that is assigned to i^{th} compute node at time slot t. $NJ(t)$ is a new array of scheduling plan at time slot t after shuffling the element of $J(t)$.

Let $JS(t)$ be the list (set) of jobs planed for time slot t associated to $J(t)$. Then

$JS(t) - JS(t+1)$ indicates the list (set) of job finishes at time slot t

$JS(t+1) - JS(t)$ indicates the list (set) of job start at time slot $t+1$

$JS(t) \cap JS(t+1)$ indicates list (set) of a job continuing from time slot t to $t+1$

A is t to $t+1$ assignment matrix of order $maxCN$ (of zeros and ones). $A(i,j) = 1$ indicates that job at time slot t is executed at compute node i and at time slot $t+1$ is executed at compute node j.

A is a partial permutation matrix. One can construct complete permutation matrix, say C_M (which is a non singular matrix).

Let C be equal to ${C_M}^T$ then the multiplication between C_M and C is I

Thus the multiplication between A and C is PI

If A is Partial Identity (PI) matrix,

Then the job at time t will be executed at the same compute node at time slot $t+1$.

Else treat advance scheduling plan for $t+1$ time slot by multiplying $J(t+1)$ with C (one can easily verify that A for this will be PI).

An Example:

Table 3.2: Reservation allocation at time slot t and $t+1$

Job at CN / Time Slot	1	2	3	4	5
$\boldsymbol{t}$	**1**	**4**	**2**	**9**	**6**
$\boldsymbol{t+1}$	**3**	**6**	**-**	**2**	**9**

From Table 3.2 we can se that job from user 2 at time slot t is allocated at compute node number 3, and at time slot $t+1$ is allocated at compute node number 4.

$$J(t) = [1\ 4\ 2\ 9\ 6]$$

$$J(t+1) = [3\ 6 - 2\ 9]$$

$$JS(t) - JS(t+1) = \{1,4\}$$

$$JS(t+1) - JS(t) = \{3\}$$

$$JS(t) \cap JS(t+1) = \{2,6,9\}$$

Construct the assignment matrix A

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

if ($J(t)(i) = J(t+1)\ (j)$) then

$A\ (i,j) = 1$

else

$A\ (i,j) = 0$

$A\ (i,j) = 1$ indicates that job at time slot t is executed at compute node i and at time slot $t+1$ is executed at compute node j.

Construct complete permutation matrix C_M

$$C_M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Every row and every column of C_M can consist only 1 (one), the other element is zero.

Construct C be equal to ${C_M}^T$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Compute $NJ(t+1) = J(t+1) * C$ then

$$NJ(t+1) = [3 \quad 6 \quad - \quad 2 \quad 9] * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$NJ(t+1) = [3 \quad - \quad 2 \quad 9 \quad 6]$$

Table 3.3: Reservation allocation at time slot t and $t+1$ after $J(t+1)$ multiplied by matrix C

Job at CN / Time Slot	1	2	3	4	5
$\boldsymbol{t}$	**1**	**4**	**2**	**9**	**6**
$\boldsymbol{t+1}$	**3**		**2**	**9**	**6**

Table 3.3 is the new reservation allocation table with the new value of $NJ(t+1)$

Construct the A

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

A is partial identity matrix (PI), then we can say that all the job at time t will be continued at the same dedicated compute node at time $t + 1$.

3.4 Illustration

PARAMETRIC JOB: To explain how FCFS-EDS for parametric job works we introduce an example. If we have $maxCN$ compute node (physical view), let $maxCN$ is equal to 5 ($cn0 - cn4$) as physical nodes, then the number of virtual node (logical view) is also 5 ($v0 - v4$). Sequence of incoming reservation is depicted in Table 3.4, where $numCN \leq maxCN$ and $numJ$ are the number of jobs submitted by $userId$. For example the given parameters for $userId$ = 3 in the Table 3.4 implies the following: "*User 3 reserved 3 time slots at time slot 12 up to 14, needed 1 compute node for 1 independent job, and cannot be delayed/shifted. ($t_{es} = t_{ls} = 12, t_e = 3, numCN = 1, numJ = 1$)*".

Table 3.4: Parameters of parametric job reservation request

$userId$	t_{es}	t_{ls}	t_e	$numCN$	$numJ$
1	11	11	1	2	2
2	11	11	3	1	1
3	12	12	3	1	1
4	15	16	1	1	1
5	15	15	1	2	2
6	11	11	2	1	1
7	13	13	1	1	1
8	16	16	2	1	1
9	11	11	2	1	1
10	15	15	3	1	1

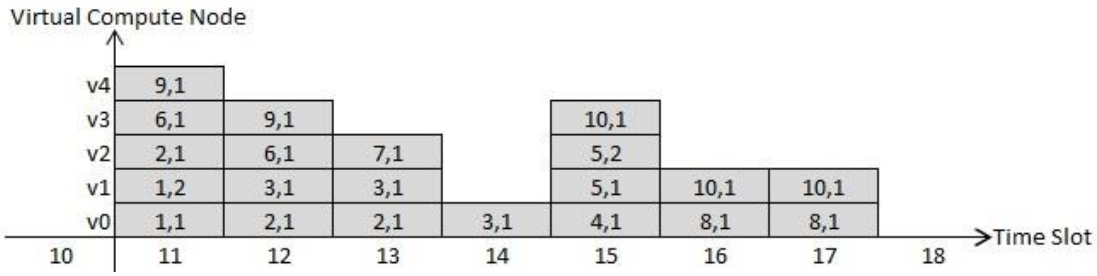

Figure 3.6: Ten reservations have been allocated (Logical View) for parametric job

The result of FCFS-EDS for parametric job is a plan shown in Figure 3.6, where x axis indicates time slot, and y axis indicates virtual compute node (Logical View). As there are 5 virtual compute nodes, they are shown along the y axis as $v0, v1, v2, v3$, and $v4$. The ten user reservations have been allocated during time slots 11 to 17. Consider $userId = 9$ from Table 3.4. The virtual computes nodes allotted to it are $v4$ at time slot 11 ($t_{es} = 11$) and $v3$ at time slot 12 as only 1 job (requiring 2 execution time slots) has been submitted by this user.

Suppose User 11 wishing to reserve 3 time slots from 12 up to 14, needs 3 compute nodes for 3 independent jobs and each job can be delayed up to time slot 14 ($t_{es} = 12, t_{ls} = 14, t_e = 3, numCN = 3, numJ = 3$). See Figure 3.7.

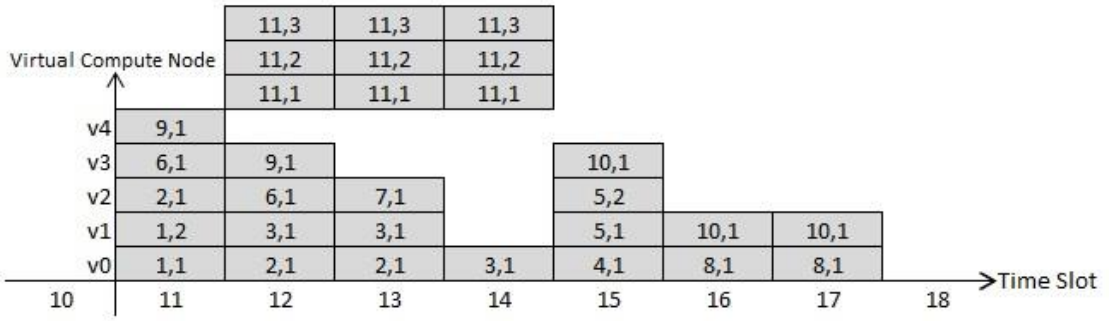

Figure 3.7: New user makes a request for parametric job

The result of FCFS-EDS for parametric job is a plan for the new incoming reservation request from user 11 which is depicted in Figure 3.8. Using conventional reservation or rigid reservation only one independent job from user 11 will be allocated and the other two will be rejected. From Figure 3.8, it is seen that the same job on different time slot is allocated on different virtual compute nodes. On

successful reservation a notification is sent to the user only once (in our approach as we are working at a logical view) whereas in other approaches it has to send every time a revision is made in the plan (P_{new}) (physical view: reassignment of physical resources) [63] [65].

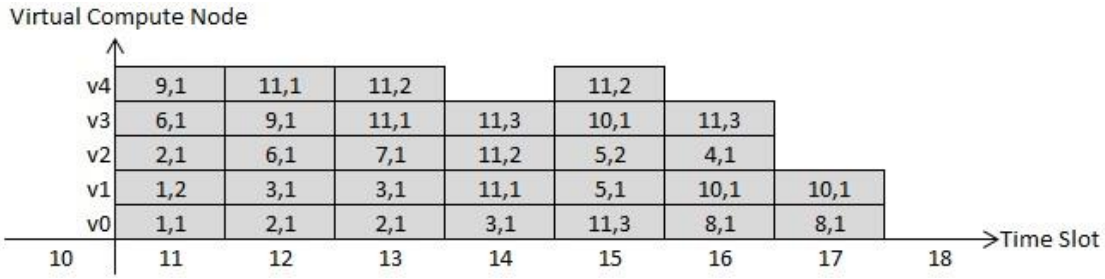

Figure 3.8: New user has been allocated using FCFS – EDS (Logical View) for parametric job

Table 3.5: Parameters of MPI job reservation request

$userId$	t_{es}	t_{ls}	t_e	$numCN$	$numJ$
1	11	11	1	2	1
2	11	11	3	1	1
3	12	12	3	1	1
4	15	16	1	1	1
5	15	15	1	2	1
6	11	11	2	1	1
7	13	13	1	1	1
8	16	16	2	1	1
9	11	11	2	1	1
10	15	15	3	1	1

MPI JOB: To explain how FCFS-EDS for MPI job works we introduce an example. If we have $maxCN$ compute node (physical view), let $maxCN$ is equal to 5 ($cn0 - cn4$) as physical nodes, then the number of virtual node (logical view) is also 5 ($v0 - v4$). Sequence of incoming reservation is depicted in Table 3.5, where $numCN \leq maxCN$ and $numJ$ are the number of jobs submitted by $userId$. For example the given

parameters for $userId$ = 3 in the Table 3.5 implies the following: "*User 3 reserved 3 time slots at time slot 12 up to 14, needed 1 compute node for 1 independent job, and cannot be delayed/shifted. ($t_{es} = t_{ls} = 12, t_e = 3, numCN = 1, numJ = 1$)*".

The result of FCFS-EDS for MPI job is a plan shown in Figure 3.9, where x axis indicates time slot, and y axis indicates virtual compute node (**Logical View**). As there are 5 virtual compute nodes, they are shown along the y axis as $v0, v1, v2, v3,$ and $v4$. The ten user reservations have been allocated during time slots 11 to 17. Consider $userId = 9$ from Table 3.5. The virtual computes nodes allotted to it are $v4$ at time slot 11 ($t_{es} = 11$) and $v3$ at time slot 12 as only 1 job (requiring 2 execution time slots) has been submitted by this user.

Suppose User 11 wishing to reserve 3 time slots from 12 up to 14, needs 2 compute nodes for 1 independent job and can be delayed up to time slot 14 ($t_{es} = 12, t_{ls} = 14, t_e = 3, numCN = 2, numJ = 1$). See Figure 3.10.

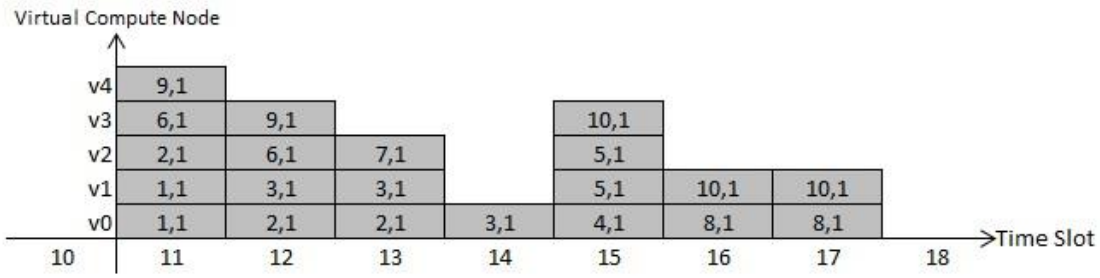

Figure 3.9: Allocation of ten MPI reservations in Logical View

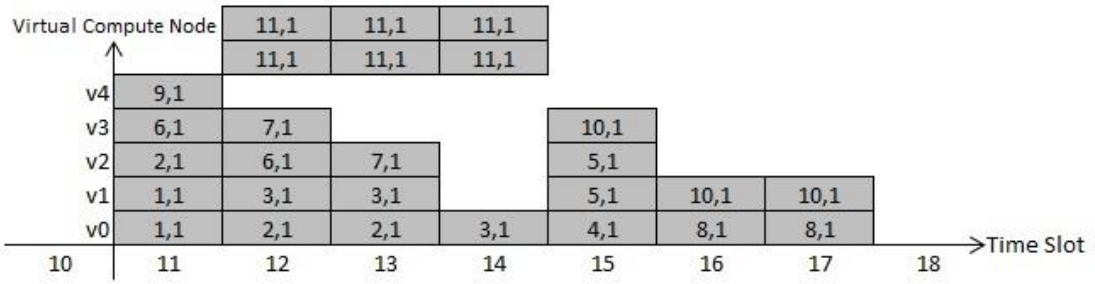

Figure 3.10: MPI reservation request from a new user

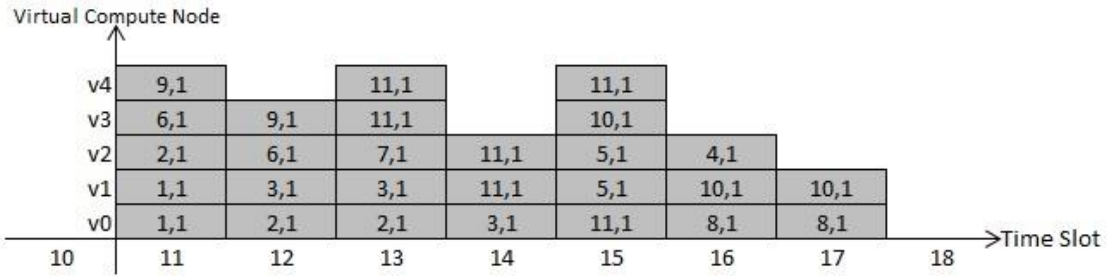

Figure 3.11: A new user has been allocated using FCFS – EDS (Logical View) for MPI job

The result of FCFS-EDS for MPI job is a plan for the new incoming reservation request from user 11 which is depicted in Figure 3.11. Using conventional reservation or rigid reservation job from user 11 will be rejected. From Figure 3.11, it is seen that the same job on different time slot is allocated on different virtual compute nodes. On successful reservation a notification is sent to the user only once (in our approach as we are working at a logical view) whereas in other approaches it has to send every time a revision is made in the plan (P_{new}) (physical view: reassignment of physical resources) [63] [65].

3.5 Mapping of the Plan (Logical View) to Actual Compute Node (Physical View)

We introduce an FCFS-EDS's Lemma to guarantee that the plan (logical view) can always be mapped into actual compute node (physical view), and once a job is started at certain compute node, then it will be executed on the same compute node for the entire time slot. Applying the FCFS-EDS's lemma to the plan of parametric job as depicted in Figure 3.12, we guarantee that all the jobs will be executed as shown in Figure 3.13 (Physical View).

Figure 3.12: Allocation/Plan for reservations request using FCFS–EDS for parametric jobs (Logical View)

Figure 3.13: Actual Compute Node Mapping (physical view) at $t_0 = 18$ for parametric jobs

Applying the FCFS-EDS's lemma to the plan of MPI job as depicted in Figure. 3.14, we guarantee that all the jobs will be executed as shown in Figure 3.15 (Physical View).

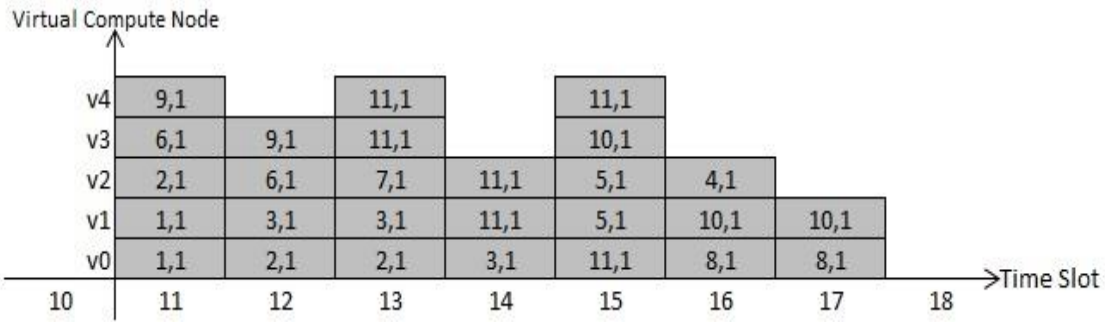

Figure 3.14: Allocation/Plan for reservation requests using FCFS–EDS for MPI jobs (Logical View)

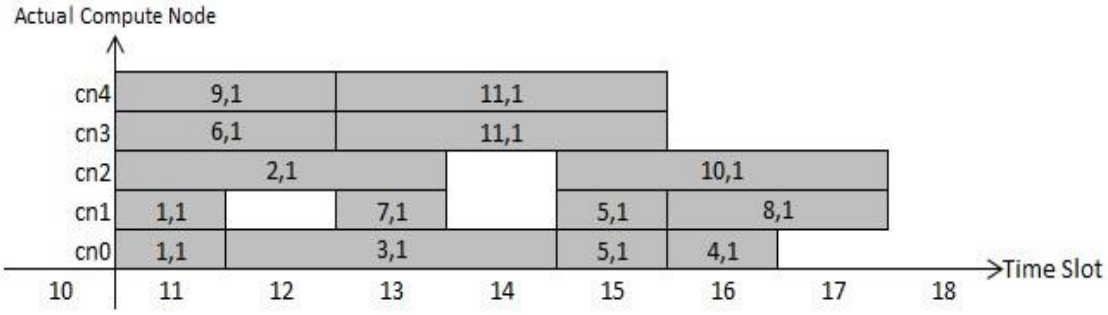

Figure 3.15: Actual Compute Node Mapping (physical view) at $t_0 = 18$ for MPI jobs

3.6 Discussion

In this chapter we have discussed our proposed Advance Planning and Reservation System. We have introduced a definition for flexible advance reservation. With that definition we have compared the parameters that have been used by other researchers in their study.

We also have discussed about our Advance Planning and Reservation System strategy namely First Come First Serve – Ejecting Base Dynamic Scheduling (FCFS-EDS). We discussed the model and algorithm of our FCFS-EDS advance planning and reservation strategy. FCFS-EDS is a scheduling strategy in a logical view. To map the plan to physical view, we introduced a new lemma (FCFS-EDS's lemma), to guarantee that once a job is started at certain compute node, then it will be executed on the same dedicated compute node for the entire time slot.

To have a better insight of our proposed strategy we have presented an illustration for both parametric and MPI jobs.

4 Data Structure for FCFS-EDS Strategy

We have proposed advance planning and reservation strategy to increase resource utilization namely First Come First Serve – Ejecting base Dynamic Scheduling (FCFS – EDS) [92]. In order to implement reliable FCFS – EDS scheduling, it is important to store information in data structure about future allocations and to provide fast access to the available information. This chapter proposes a novel data structure used by FCFS – EDS scheduling system for planning.

4.1 Introduction

There are several data structures for administering advance reservation. In general there are two types of data structure for administering advance reservation i.e. time slotted data structure and continuous data structure. In time slotted data structure each request is stored in a certain number of consecutive time slots, where in continuous data structure each request defines its own time scale. Time slotted data structure approach has the advantage of restricting the amount of data that must be stored, i.e., the memory consumption is bounded and, furthermore, it can be easily implemented [67]. The majority of current implementations in the field of advance reservations support time slotted data structure [47] [68] [69] [70] [71] [72] [73]. Consequently, our proposed data structure presented here is designed to support slotted time.

A tree-based data structure is commonly used for admission control in network bandwidth reservation [71] [74] [75]. Brown et al. [72] have proposed Calendar Queue (CalendarQ), as a priority queue for future event set problems in discrete event simulation. Qing Xiong et al. [77] have proposed a linked-list data structure for advance reservation admission control. Sulistio et al. [73] have proposed GarQ (Grid Advance Reservation Queue) for administering advance reservation in grid system. GarQ was partly influenced by Calendar Queue data structure. GarQ has buckets with a fixed δ, which represents the smallest slot duration, as with the Calendar Queue. Sulistio et al. [73] also implemented all data structures aforementioned above in grid system and compare the performance of GarQ and the other data structure. They had a result that GarQ performed better among the other data structure.

4.2 Proposed Data Structure

Existing data structures reported in the literature cannot be used for our scheduling strategy (FCFS-EDS [92] [93]) for administering advance planning. Our proposed data structure for administering advance reservation using FCFS-EDS scheduling strategy is influenced by GarQ data structure, because it has better performance among other data structure reported in the literature. We modify the GarQ so that it can handle the flexible advance planning where in GarQ can only handle the rigid reservation. This is accomplished by adding t_{es} and t_{ls} property, and omitting t_c in the data structure. The proposed data structure has buckets with a fixed time (our assumption is for 5 minutes but it can be user defined), which represents the smallest slot duration, as described in the Calendar Queue.

Our proposed data structure comes with two flavors, one for parametric job and one for MPI job. These are discussed in next sections.

4.2.1 Parametric Jobs

Our proposed data structure for parametric job can be seen in Figure 4.1, which is depicted as array based data structure. Name of the array is $tslot$. The index of the

array represents a particular time slot. Each time slot contains a list of reservations that starts in that time slot.

A Node or an Element of the $tslot$ array is a record that contains two fields (variables), i.e. $free$ field that holds the number of free virtual compute nodes available in the given time slot and the p field is a pointer to another linked list of nodes. These nodes contain the following information about a job:

$userId$: identification of the user

$jobId$: As user can submit more than one independent job, $jobId$ is used to identify them

t_{es} : the earliest start time of the job

t_{ls} : the last start time of the job

t_e : the execution time of the job

$next$: pointer to a node of reservation

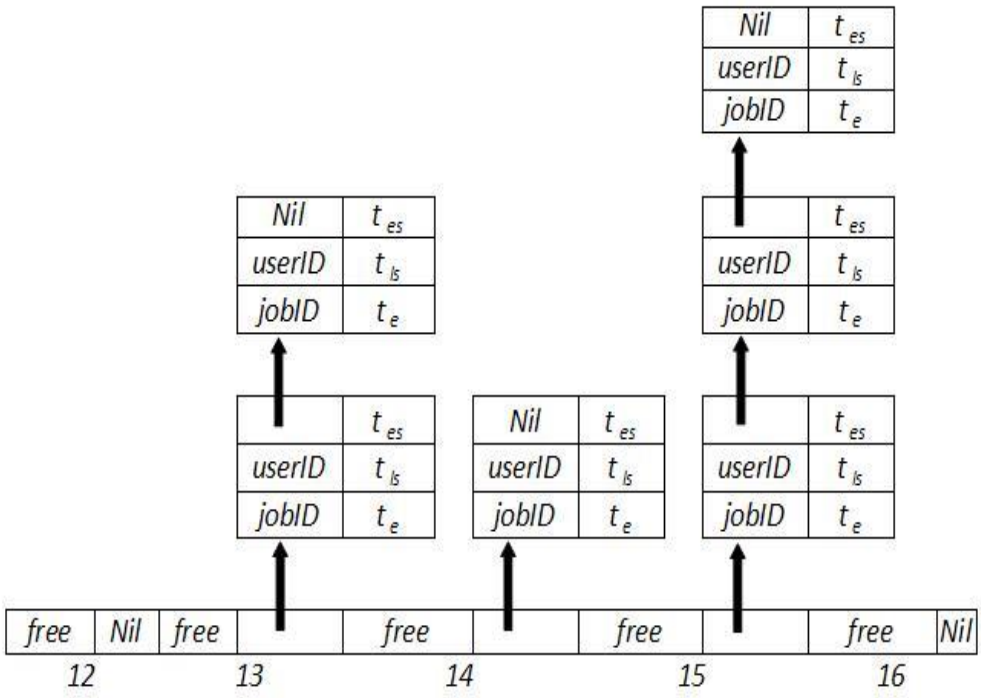

Figure 4.1: Proposed Data Structure for Parametric Job

Let us illustrate this with an example. Let us assume that we have a total number of 5 virtual compute nodes i.e., $maxCN = 5$, and a sequence of incoming reservation request from the users as depicted in Table 4.1. Consider the given parameters for $userId = 3$ in the Table 4.1. The information provided implies the following: "*User 3 has reserved 3 time slots at* $tslot$ *array, and the indexes are ranging from 12 to 14, and that the submitted job cannot be delayed or shifted because* $t_e = 3$ *and* $t_{es} = t_{ls} = 12$". Data structure that results from storing all the reservation requests of Table 4.1 can be seen in Figure 4.2. As we see in Figure 4.2 there are five jobs that start at timeslot 11, three jobs that start at timeslot 12, and so on. In the node of reservation, $next$ is equal to Nil if it doesn't point to any node of reservation. In the $tslot$ array, pointer p is equal to Nil if it doesn't point to any node of reservation.

Table 4.1: Parameters of reservation request

$userID$	$jobID$	t_{es}	t_{ls}	t_e
1	1	11	11	1
1	2	11	11	1
2	1	11	11	3
3	1	12	12	3
4	1	15	16	1
5	1	15	15	1
5	2	15	15	1
6	1	11	11	2
7	1	13	13	1
8	1	16	16	2
9	1	11	11	2
10	1	15	15	3

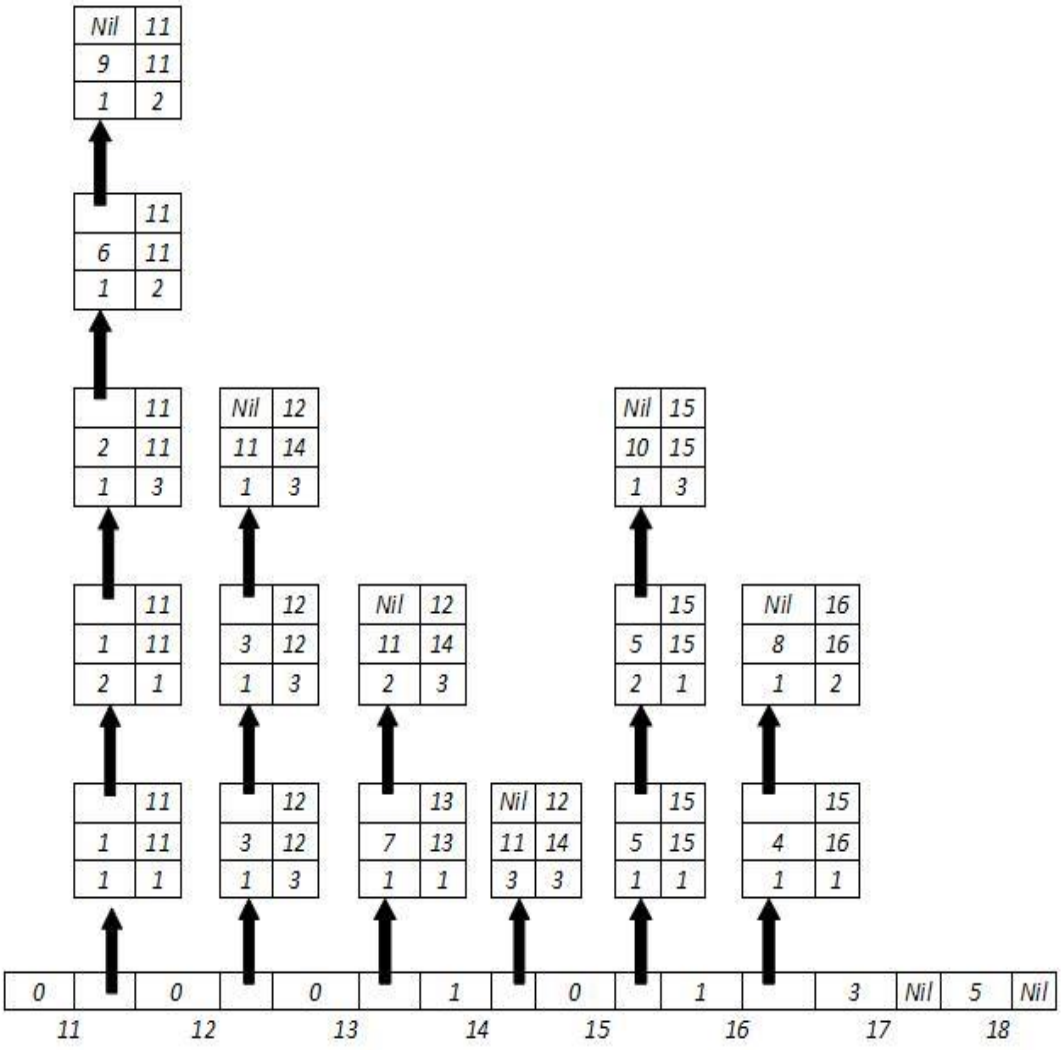

Figure 4.2: Resultant Data Structure for storing reservation requests of Table 4.1

4.2.2 MPI Jobs

Our proposed data structure for MPI job can be seen in Figure 4.3, which is depicted as array based data structure. Name of the array is $tslot$. The index of the array represents a particular time slot. Each time slot contains a list of reservations that starts in that time slot.

A Node or an Element of the $tslot$ array is a record that contains two fields (variables), i.e. $free$ field that holds the number of free virtual compute nodes

available in the given time slot and the p field is a pointer to another linked list of nodes. These nodes contain the following information about a job:

$userId$: identification of the user
$jobId$: As user can submit more than one independent job, $jobId$ can identify them
t_{es} : the earliest start time that the job can be started
t_{ls} : the last start time that the job can be started
t_e : the execution time of the job
$numCN$: number of compute node needed
$next$: pointer to a node of reservation

Let us illustrate this with an example. Let us assume that we have a total number of 5 virtual compute nodes i.e., $maxCN = 5$, and a sequence of incoming reservation request from the users as depicted in Table 4.2. Consider the given parameters for $userId = 3$ in the Table 4.2. The information provided implies the following: "*User 3 has reserved 3 time slots at* $tslot$ *array, and the indexes are ranging from 12 to 14, and that the submitted job cannot be delayed or shifted because* $t_e = 3$ *and* $t_{es} = t_{ls} = 12$, *and compute node needed is* 1". Data structure that results from storing all the reservation requests of Table 4.2 can be seen in Figure 4.4. As we see in Figure 4.4 there are four jobs that start at timeslot 11, three jobs that start at timeslot 12, and so on. In the node of reservation, $next$ is equal to Nil if it doesn't point to any node of reservation. In the $tslot$ array, pointer p is equal to Nil if it doesn't point to any node of reservation.

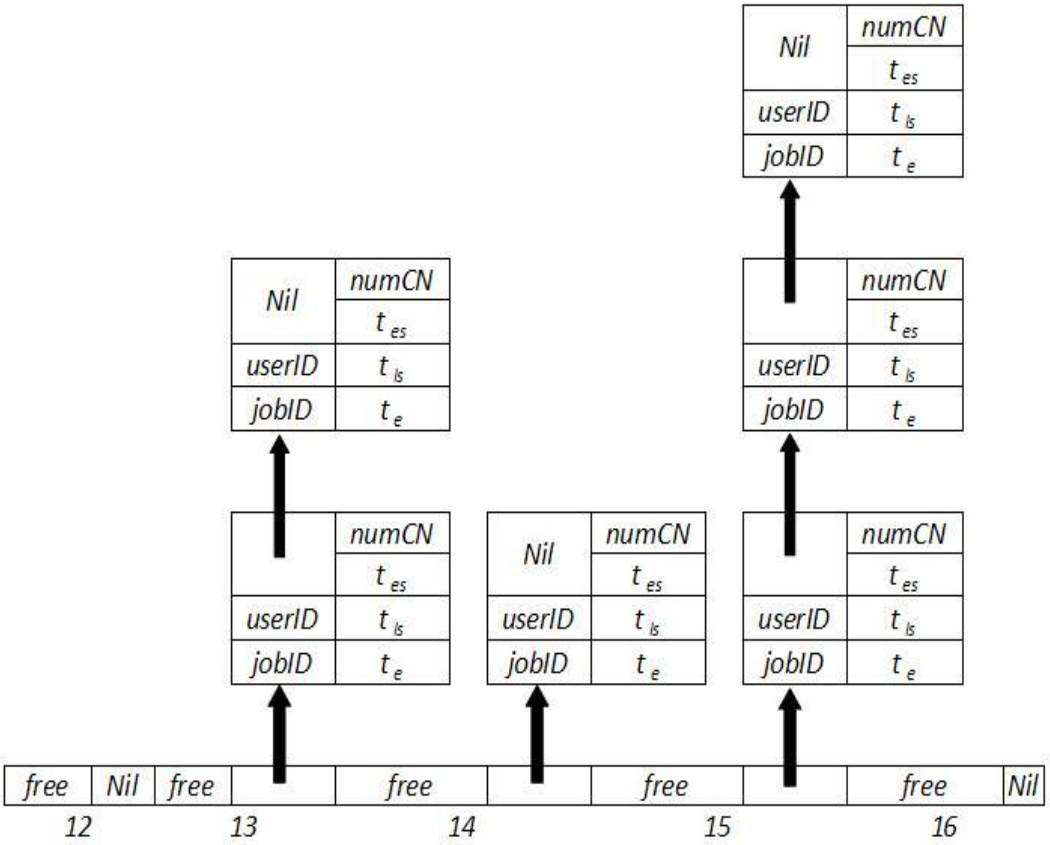

Figure 4.3: Proposed Data Structure for MPI job

Table 4.2: Parameters of reservation request of MPI job

userId	***jobId***	t_{es}	t_{ls}	t_e	***numCN***
1	1	11	11	1	2
2	1	11	11	3	1
3	1	12	12	3	1
4	1	15	16	1	1
5	1	15	15	1	2
6	1	11	11	2	1
7	1	13	13	1	1
8	1	16	16	2	1
9	1	11	11	2	1
10	1	15	15	3	1

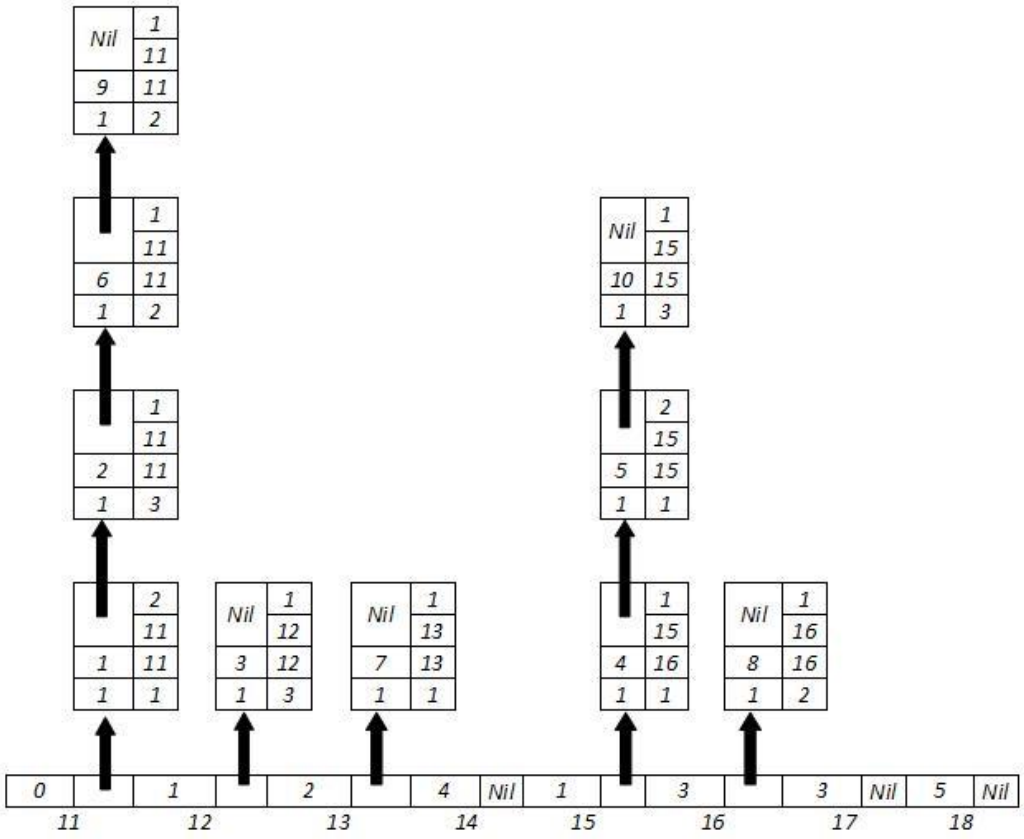

Figure 4 4: Resultant Data Structure for storing reservation requests of Table 4.2

4.3 Planning Algorithm

FCFS-EDS (First Come First Serve - Ejecting base Dynamic Scheduling) strategy takes an advantage of shifting earlier reservations made (subject to given flexible constraints) to make room for new incoming reservation request. However the planning for reservation in our proposed model provides a logical view as against the physical view reported in the literature.

4.3.1 Parametric Job

4.3.1.1 Searching an Available Virtual Node for Parametric job

Algorithm 4.1 shows an algorithm of FCFS-EDS for searching an available virtual node for parametric job. Let us assume that earlier "$n-1$" reservation requests made

to FCFS-EDS, specified by the following parameters: $userId, jobId,\ t_{es}, t_{ls}, t_e$, have been successfully planned. Definition of variables is explained in lines 4-9. Lines 10–14 initialize the variables.

Incoming "n^{th}" reservation request is scheduled based on first fit strategy (iteration of lines 16-25). It tries to search for resources within the given constraints (t_{es} and t_{ls}) and without disturbing plan for previous "$n-1$" reservation requests that were made. Let us call this as plan P_{old}. If within the given flexible constraints the search fails to allocate resources the algorithm tries to move around previous "$n-1$" reservations to accommodate "n^{th}" reservation request (lines 31-44). If the resources are found then the search is declared successful and the algorithm outputs a new plan P_{new} that depicts "n" reservation requests as a logical view. If the search is a failure then the "$n-1$" reservation request plan has to be restored to its previous state i.e. P_{old}.

Algorithm 4.1: Searching available virtual node for parametric job

```
1 Function searchAndAlloc(userId, jobId, tes, tls, te : integer) → boolean
2 //search and allocate job with given tes, tls, te
3  Dictionary :
4  ts : integer /*start time of the job*/
5  tc : integer /*finish time of the job*/
6  min : integer  /*minimum free nodes within interval  ts - tc*/
7  t   : integer  /*timeslot of minimum available node between ts to tc*/
8  tr  : integer  /*relax time, length between of tes and tls*/
9  relax : integer /*different between ts and tes time (ts - tes + 1)*/
Algorithm :
10 tr ← tls – tes
11 succeed ← false
12 ts ← tes
13 tc ← tes + te – 1
14 relax ←  ts - tes
15
16 while (!succeed AND (relax≤tr)) do  /*searching by first fit strategy*/
17    /*searching minimum free node between ts to tc*/
18    min,t ← minFreeNode(ts, tc)
19      if(min > 0) then
20        add(userId, jobId, tes, ts, tls, te)
21        succeed ← true
22      else
23        ts ← t + 1
24        tc ← ts + te - 1
25        relax ← ts - tes
26 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
27
```

```
28 ts ← tes
29 tc ← tes + te – 1
30 relax ←  ts - tes
31 while (!succeed AND (relax≤tr)) do
32    /*searching minimum free node between ts to tc*/
33    min,t ← minFreeNode(ts, tc);
34    if(min > 0) then
35      /*push or plan the job to data structure and update free node
36      between ts to tc*/
37      add(userId, jobId, tes, ts, tls, te)
38      succeed ← true
39    else
40      /*try to shift a job that start at t time slot*/
41      if(!shiftNode(t)) then //can't be shifted, move ts to t+1
42        ts ← t + 1
43        tc ← ts + te - 1
44        relax ← ts - tes
45 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
46 if (!succeed)
47   putBackAllShiftedJob()
48 return succeed
```

4.3.1.2 Adding a Parametric Job

Algorithm 4.2 shows an algorithm for allocating/adding the reservation for parametric job in the data structure. Here new reservation is put in an ascending order by $userId$. If a user ($userId$) has more than one reservation then new reservation is put in an ascending order by $jobId$. Definition of variables is explained in lines 5-7. Lines 9–10 initialize the variables.

There are 4 possible cases to add the new reservation in data structure:

1. The list of reservation in ${t_s}^{th}$ element of timeslot array is empty. In this case, add the new reservation as the first node of the list of reservation (insert first). This addition is done in line 13.

2. First node of the list of reservation has $userId$ which is bigger than the incoming $userId$. In this case add the new reservation as the first node of the list of reservation (insert first). This addition is done in lines 15 and 16.

3. The first element of the list of reservation has the same $userId$ with the incoming $userId$ and $jobId$ of the first node of list of reservation is bigger than the incoming $jobId$. Add the new reservation as the first component of the list of

reservation (insert first). This addition is done in lines 18 and 19

4. In this case the new reservation will be inserted in the middle or in the last of reservation. pn is a pointer to a reservation node, this pointer is used for traversing the list of reservation. While traversing the list of reservation there are three possibilities that can happen:

   a. All nodes in the reservation list have smaller $userId$ than the incoming reservation $userId$. In this case, add the new reservation as the last node of the reservation list (insert last). This addition is done in line 26.

   b. There is a node in the reservation list which has bigger $userId$ than the incoming $userId$. In this case add the new reservation right before the node which has the bigger $userId$ than the incoming $userId$. This addition is done in lines 28-29

   c. There is no node in the reservation list which has the same $userId$ with the incoming $userId$. In this case, pn traverse in among the nodes which have the same $userId$ with the incoming reservation $userId$. While traversing, there are three possibilities that can happen:

      i. It reaches the last node of reservation list which $jobId$ is smaller than $jobId$ of the incoming reservation. In this case, add the new reservation as the last node of the reservation list (insert last). This addition is done in line 34

      ii. The last node of traversal (name it ln node) has smaller $jobId$ than the incoming reservation $jobId$. In this case, add the new reservation directly after node ln. This addition is done in lines 36-37.

      iii. It finds the node (name it fn node) which has the $jobId$ bigger than the incoming reservation $jobId$. In this case, add the new reservation directly before node fn. This addition is done in lines 39-40.

Updating the free field of *tslot* array is done by lines 42-43

Algorithm 4.2: Adding a reservation for parametric job

```
1  Procedure add(userId, jobId, tes, ts, tls, te : integer)
2  //allocate a reservation with userId, jobId, eStartTime=tes,
lStartTime=tls,
3  //execTime=te at ts^th element of the timeslot array (tslot)
4  Dictionary :
5  tc : integer /*finish time of the job*/
6  n : node /*record of reserevation*/
7  pn : pointer to node
8  Algorithm :
9  tc ← ts + te - 1
10 n ← new node(userId, jobId, tes, tls, te, nill)
11
12 if (tslot[ts].p = nil) then
13    tslot[ts].p ← n     //insert first
14 else if (tslot[ts].p.userId > n.userId) then
15    n.next ← tslot[ts].p
16    tslot[ts].p ← n     //insert first
17 else if (tslot[ts].p.userId = n.userId AND tslot[ts].p.jobId > n.jobId)
then
18    n.next ← tslot[ts].p
19    tslot[ts].p ← n     //insert first
20 else
21    pn ← tslot[ts].p
22    while (pn.next!=nil AND pn.next.userId < n.userId) do
23       pn ← pn.next
24    //here pn.next.userId > n.userId or pn.next.userId = n.user.Id or
pn.next = nill
25    if (pn.next = nil) then
26       pn.next ← n        //insert last
27    else if (pn.next.userId > n.userId)
28       n.next ← pn.next   //insert new userId
29       pn.next ← n
30    else //the same userId is found or pn.next.userId = n.user.id
31       while(pn.next!=nil AND pn.next.userId=n.userId AND
pn.next.jobId<n.jobId) do
32          pn ← pn.next
33       if(pn.next = nil) then
34          pn.next ← n    //insert last
35       else if(pn.next.userId != n.userId) then
36          n.next ← pn.next   //insert last for old userId
37          pn.next ← n
38       else if (pn.next.jobId > n.jobId) then
39          n.next ← pn.next
40          pn.next ← n    //insert old userId
41
42 for(int i=ts; i<=tc; i++)
43    tslot[i].free ← tslot[i].free -1  //update a free node
```

4.3.1.3 Deleting a Parametric Job

Deleting an existing reservation for parametric job is shown in Algorithm 4.3. Definition of variables is explained in lines 4-6. Lines 8 and 9 initialize the variables. In deleting the existing reservation there are two possible cases that can happen:

1. The first node of reservation list at $t_s{}^{th}$ element of timeslot array has the same $userId$ and $jobId$ with the node that will be deleted. In this case delete the first node (delete first). This is done in lines 11 and 12.

2. Traverse the reservation list at $t_s{}^{th}$ element of timeslot array until finding the node (name it dn) which has the same $userId$ and $jobId$ with the node that will be deleted. In this case delete the dn node. This is done in lines 16-18.

Updating the free field of $tslot$ array is done by lines 20-21.

Algorithm 4.3: Deleting a reservation for parametric job

```
1  Procedure delete(userId, jobId, ts : integer)
2  //delete a reservation with userId, jobId
3  Dictionary :
4  tc : integer /*finish time of the job*/
5  pn : pointer to node
6  pdel : pointer to node
7  Algorithm :
8  tc ← ts + te – 1
9  pn ← tslot[ts].p
10 if(pn.userId = userId AND pn.jobId = job.Id) //delete first
11    tslot[ts].p = pn.next
12    delete pn //delete pn from memory
13 while (pn.next.userId != userId OR pn.next.jobId != jobId) do
14     pn ← pn.next
15
16 pdel ← pn.next
17 pn.next ← pdel.next
18 delete pdel
19
20 for(int i=ts; i<=tc; i++)
21    tslot[i].free ← tslot[i].free -1  //update a free node
```

4.3.2 MPI Job

4.3.2.1 Searching an Available Virtual Nodes for MPI Job

Algorithm 4.4 shows an algorithm of FCFS-EDS for searching available nodes for MPI job. Let us assume that earlier "$n-1$" reservation requests made to FCFS-EDS, specified by the following parameters: t_{es}, t_{ls}, t_e, $userId$, $jobId$, $numCN$ have been successfully scheduled. Definition of variables is explained in lines 4-9. Lines 10–14 initialize the variables.

Incoming "n^{th}" reservation request is scheduled based on first fit strategy (iteration of lines 16-25). It tries to search for resources within the given constraints (t_{es}, t_{ls} and $numCN$) and without disturbing plan for previous "$n-1$" reservation requests that were made. Let us call this as plan P_{old}. If within the given flexible constraints the search fails to allocate resources the algorithm tries to move around previous "$n-1$" reservations to accommodate "n^{th}" reservation request (lines 31-44). If the resources are found then the search is declared successful and the algorithm outputs a new plan P_{new} that depicts "n" reservation requests as a logical view. If the search is a failure then the "$n-1$" reservation request plan has to be restored to its previous state i.e. P_{old}.

Algorithm 4.4: Searching available virtual nodes for MPI job

```
1 Function searchAndAlloc(userId, jobId, tes, tls, te, numCN : integer) →
boolean
2 //search and allocate job with given tes, tls, te, numCN
3  Dictionary :
4  ts : integer /*start time of the job*/
5  tc : integer /*finish time of the job*/
6  min : integer  /*minimum free nodes within interval ts - tc*/
7  t   : integer  /*timeslot of minimum available node between ts to tc*/
8  tr  : integer  /*relax time, length between of tes and tls*/
9  relax : integer /*different between ts and tes time (ts - tes + 1)*/
Algorithm :
10 tr ← tls – tes
11 succeed ← false
12 ts ← tes
13 tc ← tes + te – 1
14 relax ←  ts - tes
15
16 while (!succeed AND (relax≤tr)) do  /*searching by first fit strategy*/
17    /*searching minimum free node between ts to tc*/
```

```
18    min,t ← minFreeNode(ts, tc)
19      if(min ≥ numCN) then
20        add(userId, jobId, tes, ts, tls, te, numCN)
21        succeed ← true
22      else
23        ts ← t + 1
24        tc ← ts + te - 1
25        relax ← ts - tes
26 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
27
28 ts ← tes
29 tc ← tes + te – 1
30 relax ←  ts - tes
31 while (!succeed AND (relax≤tr)) do
32    /*searching minimum free node between ts to tc*/
33    min,t ← minFreeNode(ts, tc);
34    if(min ≥ numCN) then
35      /*push or plan the job to data structure and update free node
36      between ts to tc*/
37      add(userId, jobId, tes, ts, tls, te, numCN)
38      succeed ← true
39    else
40      /*try to shift a job that start at t time slot*/
41      if(!shiftNode(t, numCN-min)) then //can't be shifted, ts equals t+1
42        ts ← t + 1
43        tc ← ts + te - 1
44        relax ← ts - tes
45 /*end while, the state is succeed=true or succeed=false (relax > tr)*/
46 if (!succeed)
47   putBackAllShiftedJob()
48 return succeed
```

4.3.2.2 Adding an MPI Job

Algorithm 4.5 shows an algorithm for allocating/adding the reservation for MPI job in the data structure. Here new reservation is put in an ascending order by $userId$. If a user ($userId$) has more than one reservation then new reservation is put in ascending order by $jobId$. Definition of variables is explained in lines 5-7. Lines 9–10 initialize the variables.

There are 4 possible cases to add the new reservation in data structure:

1. The list of reservation in $t_s{}^{th}$ element of timeslot array is empty. In this case, add the new reservation as the first node of the list of reservation (insert first). This addition is done in line 13.

2. First node of the list of reservation has $userId$ which is bigger than the incoming $userId$. In this case add the new reservation as the first node of the list of reservation (insert first). This addition is done in lines 15 and 16.

3. The first element of the list of reservation has the same $userId$ with the incoming $userId$ and $jobId$ of the first node of list of reservation is bigger than the incoming $jobId$. Add the new reservation as the first component of the list of reservation (insert first). This addition is done in lines 18 and 19

4. In this case the new reservation will be inserted in the middle or in the last of reservation. pn is a pointer to a reservation node, this pointer is used for traversing the list of reservation. While traversing the list of reservation there are three possibilities that can happen:

    a. All nodes in the reservation list have smaller $userId$ than the incoming reservation $userId$. In this case, add the new reservation as the last node of the reservation list (insert last). This addition is done in line 26.

    b. There is no node in the reservation list with the bigger $userId$ then the incoming $userId$. In this case add the new reservation right before the node which has the bigger $userId$ than the incoming $userId$. This addition is done in lines 28-29

    c. There is a node in the reservation list which has the same $userId$ with the incoming $userId$. In this case, pn traverse in among the nodes which have the same $userId$ with the incoming reservation $userId$. While traversing, there are three possibilities that can happen:

        i. It reaches the last node of reservation list which $jobId$ is smaller than $jobId$ of the incoming reservation. In this case, add the new reservation as the last node of the reservation list (insert last). This addition is done in line 34

        ii. The last node of traversal (name it ln node) has smaller $jobId$ than the

incoming reservation $jobId$. In this case, add the new reservation directly after node ln. This addition is done in lines 36-37.

iii. It finds the node (name it fn node) which has the $jobId$ bigger than the incoming reservation $jobId$. In this case, add the new reservation directly before node fn. This addition is done in lines 39-40.

Updating the free field of $tslot$ array is done by lines 42-43.

Algorithm 4.5: Adding a reservation for MPI job

```
1  Procedure allocate(userId, jobId, tes, ts, tls, te, numCN : integer)
2  //allocate a reservation with userId, jobId, eStartTime=tes,
lStartTime=tls,
3  //execTime=te at ts^th element of the timeslot array (tslot)
4  Dictionary :
5  tc : integer /*finish time of the job*/
6  n : node /*record of reserevation*/
7  pn : pointer to node
8  Algorithm :
9  tc ← ts + te - 1
10 n ← new node(userId, jobId, tes, tls, te, nill)
11
12 if (tslot[ts].p = nil) then
13    tslot[ts].p ← n     //insert first
14 else if (tslot[ts].p.userId > n.userId) then
15    n.next ← tslot[ts].p
16    tslot[ts].p ← n     //insert first
17 else if (tslot[ts].p.userId = n.userId AND tslot[ts].p.jobId > n.jobId)
then
18    n.next ← tslot[ts].p
19    tslot[ts].p ← n     //insert first
20 else
21    pn ← tslot[ts].p
22    while (pn.next!=nil AND pn.next.userId < n.userId) do
23       pn ← pn.next
24    //here pn.userId > n.userId or pn.userId = n.user.Id or pn.next = nill
25    if (pn.next = nil) then
26       pn.next ← n        //insert last
27    else if (pn.next.userId > n.userId)
28       n.next ← pn.next   //insert new userId
29       pn.next ← n
30    else //the same userId is found or pn.next.userId = n.user.id
31       while(pn.next!=nil AND pn.next.userId=n.userId AND
pn.next.jobId<n.jobId) do
32          pn ← pn.next
33       if(pn.next = nil) then
34          pn.next ← n    //insert last
35       else if(pn.next.userId != n.userId) then
36          n.next ← pn.next   //insert last for old userId
```

```
37          pn.next ← n
38       else if (pn.next.jobId > n.jobId) then
39          n.next ← pn.next
40          pn.next ← n    //insert old userId
41
42 for(int i=ts; i<=tc; i++)
43    tslot[i].free ← tslot[i].free -numCN  //update a free node
```

4.3.2.3 Deleting an MPI Job

Deleting an existing reservation is shown in Algorithm 4.6. Definition of variables is explained in lines 4-6. Lines 8 and 9 initialize the variables. In deleting the existing reservation there are two possible cases that can happen:

3. The first node of reservation list at $t_s{}^{th}$ element of timeslot array has the same $userId$ and $jobId$ with the node that will be deleted. In this case delete the first node (delete first). This is done in lines 11 and 12.

4. Traverse the reservation list at $t_s{}^{th}$ element of timeslot array until finding the node (name it dn) which has the same $userId$ and $jobId$ with the node that will be deleted. In this case delete the dn node. This is done in lines 16-18.

Updating the free field of $tslot$ array is done by lines 20-21.

Algorithm 4.6: Deleting a reservation for MPI job

```
1  Procedure delete(userId, jobId, ts, numCN : integer)
2  //delete a reservation with userId, jobId
3  Dictionary :
4  tc : integer /*finish time of the job*/
5  pn : pointer to node
6  pdel : pointer to node
7  Algorithm :
8  tc ← ts + te - 1
9  pn ← tslot[ts].p
10 if(pn.userId = userId AND pn.jobId = job.Id) //delete first
11    tslot[ts].p = pn.next
12    delete pn //delete pn from memory
13 while (pn.next.userId != userId OR pn.next.jobId != jobId) do
14     pn ← pn.next
15
16 pdel ← pn.next
17 pn.next ← pdel.next
18 delete pdel
19
```

```
20 for(int i=ts; i<=tc; i++)
21    tslot[i].free ← tslot[i].free -numCN  //update a free node
```

4.3.3 Complexity Analysis

The efficiency of the algorithm can be used to determine the best algorithm to solve the problem among several algorithms. The efficiency of the algorithm can be measured by the execution time (time needed) of the algorithm and space needed by the algorithm (empirical approach). Measuring the time needed of the algorithm by counting the real time consumption (in second) is not the correct way because of two reasons: first, every computer with different architecture has different machine language, hence time for each operation between different computers is not the same. Second, Different compiler for language programming has different machine code, hence every operation between different compilers is not the same.

Table 4.3: Time asymptotic complexity

Algorithm	Time complexity
Search Complexity for parametric job	$O(t_e.t_r)$
Add Complexity for parametric job	$O(n+t_e)$
Delete Complexity for parametric job	$O(n+t_e)$
Search Complexity for MPI job	$O(t_e.t_r)$
Add Complexity for MPI job	$O(n+t_e)$
Delete Complexity for MPI job	$O(n+t_e)$

Abstract model for measuring time/space must be independent from any computers or any compiler. Measurement used to explain the abstract model for measuring the time/space complexity is algorithm complexity. Time complexity $T(n)$ is the number of steps taken in the algorithm as a function of the input size (n). Space

complexity $S(n)$ is the amount of space used in the algorithm as a function of the input size (n). The "Big O" notation is used for time asymptotic complexity, and $O(n)$ is an asymptotic upper-bound for $T(n)$.

Our proposed data structure has time asymptotic complexity as depicted in Table 4.3. where t_e is execution time, t_r is the relax time (the earliest start time – the latest start time) and n is the number of reservation in the list in for each timeslot.

4.4 Discussion

In this chapter we have discussed our proposed data structure for our advance planning and reservation scheduling strategy namely FCFS-EDS. We have given the example how this data structure can store the advance reservation. Algorithm for searching for a free slot, adding and deleting reservation for parametric and MPI job have also been presented here. Finally we also have presented the algorithm complexity of those proposed algorithm.

5 Revenue Management System

In this chapter we discuss about revenue management system. The goal of Revenue Management System is to maximize the revenue. Revenue management strategy can be done by doing market segmentation and price discrimination. We will show that revenue management is suitable for increasing revenue of resource provider in grid system, as computing powers can be considered non-renewable (expiry nature), has a fixed amount of capacity and a price-sensitive customer can be segmented.

5.1 Introduction

In grid system, the resource owners (producer) and resource users (consumers) have different objectives, strategies, and supply-and-demand patterns. Resource owners and resource users are geographically distributed in different time zones. The most commonly used approaches for managing such complex environments are driven by system-centric and user-centric policies [42]. System centric is a traditional approach for a resource management that attempts to optimize system-wide measure of performance. Also system centric is usually used in managing resources in single administrative domains. On the other hand, User-centric approaches focus on delivering maximum utility of the resources to the users based on their QoS requirements, i.e., an assurance of performance based on the criteria that the user thinks important such as the deadline by which his jobs have to be completed. Implementing QoS requires a system of rewards and penalties and, hence, typically a user-centric approach is driven by economic principles.

Several studies [43] [44] [45] [53] [58] [59] [63] [65] [87] have been done to improve scheduling and handling of reservations in grid local resource manager using different techniques. However, a simple pricing model to determine the usage cost of each reservation has been provided by [43] [44] [45]. To increase incentives or profits, resources provider might need to deploy a more complex pricing model.

5.2 Revenue Management System Strategy

Revenue Management is a process, for capacity-constrained industries, where the goal is to maximize revenue by allocating the right product to the right customers at the right price. The application of Revenue Management in capacity-constrained industries has some common characteristics, which are necessarily required, to successfully apply in grid system. These characteristics are [94] [95] [96]:

1. The product is non-renewable. We cannot store the product for future use if at a particular time this product is not sold. If the product is not sold then the product become unavailable or it ages. Example includes seat of aeroplane, and computing resource in grid computing. A simple way to treat this non-renewability is by making capacity a time-dependent quantity [97]. The usage of computing resources on grid computing system is limited to a point of time. For example, a user can use computing resources from 10:00 AM to 11:00 PM. If the user does not consume these computing resources in this time frame, they are wasted as the resources are not utilized by any other user.

2. Fixed amount capacity. We have a fixed amount capacity of product to be sold. Adding an additional product by buying a new one will incur high additional fixed costs and increasing maintenance cost. However, an increment in selling an additional unit of product to the customer causes only low variable costs. These characteristics are common for products like grid computing resource, where resource provider has a limited or fixed amount of computing resource.

3. The possibility of segmenting price-sensitive customers. The common mechanism to segment the customer in revenue management is the time of

purchase. The less price sensitive customers will usually wait until the last minute to make a reservation. On the other hand customers who make their reservation early on are generally more price sensitive. If offering discount prices and limiting the number who can buy at a discount price is impossible, then the situation may not be suitable for revenue management.

From the above characteristics, revenue management is suitable for increasing revenue of resource provider in grid system, as computing powers can be considered non-renewable, has a fixed amount of capacity and a price-sensitive customer can be segmented.

5.2.1 Customers Segmentation and Price Differentiation

In segmenting price sensitive customer, revenue management in grid system can adopt the way how airline industries segment their price sensitive customers. The airline offers their customers different price structure or classes based on when their customer book their flights prior to the departure time. For example, a customer that books a flight three weeks before the departure time can be identified as a leisure or budget customer. From historical data, the airline knows that, leisure customers who book three weeks before a departure time are more flexible to change and more price sensitive than business customers who book one day before a departure time. In this case, the airline can sell a lower price to leisure customers as compared to business customers for seats in a same flight.

Similarly, demand on grid resources consists of advance reservation and immediate reservation. Customers may plan in advance to run their jobs on the Grid or they may request resources immediately. The resource provider can plan the allocation accurately and set the prices effectively, if there are more reservations especially in high-demand periods [98]. Reservation of resources in grid computing has two aspects: Firstly, by reserving resources in advance can help in increasing the utilization of resources and reduce uncertainty. Secondly, with this uncertainty nature of the reservation, the resource providers are facing a problem, whether to accept the low-paying customer in advance, although a higher-paying customer could arrive later.

In grid system we can classify users according their willingness-to-pay. The resource provider has an opportunity to exploit additional profit due to different valuation by customers of grid resources or varying willingness-to-pay. The resource provider can gain more profit by adopting Revenue Management due to more degree of heterogeneity of willingness-to-pay of users. Users with high willingness to pay are less price sensitive, and they usually come later. Users with low willingness to pay are more price sensitive and usually come early to make reservations. Therefore, in grid computing we can classify a user into three classes [95]. First Premium users, we can call as class 1 users, have a highest willingness to pay for a computing resources. Second, Business users, we can call as class 2 users, have a moderate willingness to pay for a computing resources. Third, Economy users, we can call class 3 users, have the lowest willingness to pay for a computing resources. The resource provider can set prices for the same computing resource to be: $p_1 > p_2 > p_3$, where p_1 denotes the price paid by the class 1 users, p_2 denotes the price paid by the class 2 users, and p_3 denotes the price paid by the class users. This practice is commonly known in the economics literature as price differentiation or discrimination [99].

5.2.2 Capacity Allocation

Capacity allocation is the problem of determining how many computing resources to allocate to price sensitive users (discount-price users) to book/reserve computing resources when there is a possibility of future less price sensitive users (full-price) demand. Capacity allocation is an issue for grid computing industry that has the opportunity to restrict the early discount-price booking/reservation in order to reserve computing resource capacity for later full-price reservation. If too many computing resources are allocated to lower-class, we may lose a chance to earn more revenue from accepting future booking/reservation from higher-class users. On the contrary, an insufficient quota for the lower-class users, may lead to lower revenue. Thus, finding an appropriate capacity allocation to each user class at different time periods is an important factor in revenue management.

5.2.2.1 Two Classes Capacity Allocation Problem

In two class capacity allocation problem, resource provider with fixed capacity of resources has two classes of customers: discount-price customers who reserve early and full-price customers who reserve later. Discount-price customers each pay a price $p_2 > 0$, and full-price customers each pay a higher price p_1, where $p_1 > p_2$. The two class capacity allocation problem is: How many discount-price customers (if any) should we allow to reserve? Or, Equivalently: How many computing resources should we protect for full-price customers [46]?

A protection level specifies an amount of capacity to reserve (protect) for a particular class. A booking limit is the maximum number of computing resources that may be reserved at each price class. If protection level for full-price demand is demotes by y, and booking limit for discount-price customer is denoted by b, then $C = b + y$, where C is equal to capacity of resources. If we set the discount booking limit too low, we will reject discount-price customers but the full-price customers are not enough to reserve all resource capacity. This is called spoilage, since computing resource becomes spoiled by rejecting discount-price customers. On the other hand, if we allow too many discount-price customers to reserve, we have a risk of rejecting full-price customers which give us more profit. This scenario is called dilution, where we dilute the revenue we can have from saving an additional computing resources for a high-price customers. The challenge of capacity allocation is to balance the risks of spoilage and dilution to maximize the revenue.

$F_1(x)$ is the probability that full-price demand, that is less than or equal to x and $F_2(x)$ is the probability that discount-price demand is less or equal to x. d_1 is full-price demand and d_2 is discount-price demand. Assume that we have currently set a booking limit for discount-price user to $b - 1$ and all $b - 1$ computing resources have been reserved by discount-price customer. The computing resource capacity is 20, as depicted in Figure 5.1. From Figure 5.1, we already have revenue of 9 computing resource with discount price p_2.

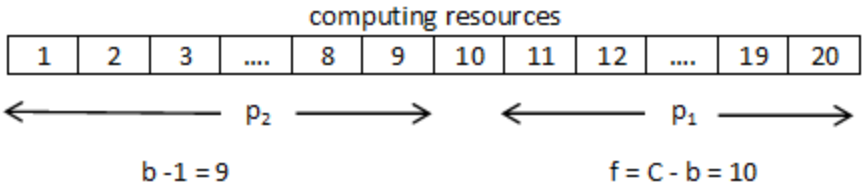

Figure 5.1: We have 20 computing resources and we have set booking limit to b-1=9

If discount-price reservation comes to reserve b^{th} computing resources, do we have to accept it or reject it. If we accept it, it means that we increased discount booking limit from $b-1$ to b. The decision we face is whether or not to increase the booking limit by one computing resource (from $b-1$ to b) or equivalently to reduce protection level by one computing resource. What would be the change in expected revenue if we increase the booking limit from $b-1$ to b? If we increase booking limit for discount-price customer/demand (d_2) from $b-1$ to b then there are several probabilities will happen:

- If discount-price customer/demand (d_2) is less than or equal to $b-1$ $(d_2 \leq b-1)$, there is no effect in revenue. The probability that this event might happen is $F_2(b-1)$.

- If discount-price customer/demand (d_2) is greater than $b-1$ $(d_2 > b-1)$, (the probability that this event might happen is $(1 - F_2(b-1))$, then we have to see the full-price customer/demand (d_1)

    - If full-price customer/demand is less or equal to $C-b$ $(d_2 \leq C-b)$, we gain revenue p_2 as discount-price customer/demand reserve computing resource number 10. The probability of this event to happen is $F_1(C-b)$

    - If full-price customer/demand is greater than $C-b$ $(d_1 > C-b)$, we lose revenue of p_1 because we because we give computing resource number 10 to discount-price demand, and we gain p_2. Overall we lose $p_1 - p_2$, or we gain $p_2 - p_1$. The probability of this event to happen is $(1 - F_1(C-b))$

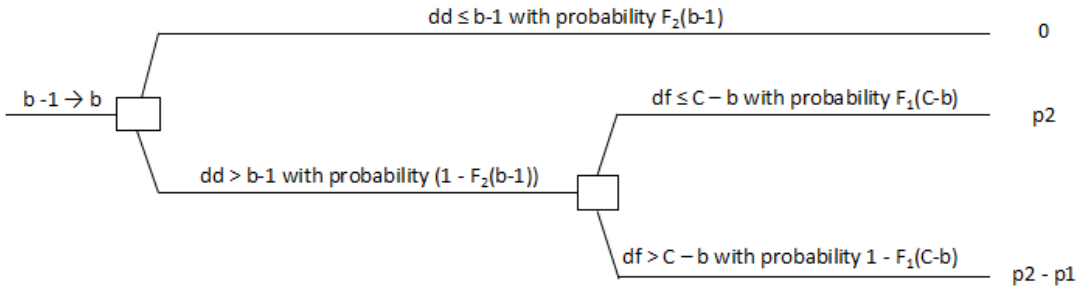

Figure 5.2: Decision Tree of two-class capacity allocation problem

The scenario above is depicted in Figure 5.2. The expected change in revenue (Er) from changing b-1 to b is the probability-weighted sum of possibilities above

$$Er = F_2(b-1) * 0 + [1 - F_2(b-1)] * \{F_1(C-b) * p_2 + (1 - F_1(C-b)) * (p_2 - p_1)\}$$

$$Er = 1 - F_2(b-1) * \{F_1(C-b) * p_2 + (1 - F_1(C-b)) * p_2 - (1 - F_1(C-b)) * p_1\}$$

$$Er = 1 - F_2(b-1) * \{(F_1(C-b) + 1 - F_1(C-b)) * p_2 - (1 - F_1(C-b)) * p_1\}$$

$$Er = 1 - F_2(b-1) * \{p_2 - (1 - F_1(C-b)) * p_1\} \quad (1)$$

We can see that revenue change is not related to $[1 - F_2(b-1)]$ because this value is always positive. So the key is $p_2 - (1 - F_1(C-b)) * p_1$, if it is positive we should increase the booking limit to b, if it is negative we should not increase the booking limit to b. As aforementioned above that the challenge of capacity allocation is to balance the risks of spoilage and dilution to maximize the revenue what we have mention, which means that the expected change revenue is zero. So we have

$$p_2 - (1 - F_1(C-b)) * p_1 = 0$$

$$1 - F_1(C-b) = \frac{p_2}{p_1}$$

$$F_1(C-b) = 1 - \frac{p_2}{p_1} \quad (2)$$

$C - b = y$, is the protection level for full-price customer. A protection level specifies an amount of capacity to reserve (protect) for a particular class.

Littlewood's rule [100] states that, in a two-price environment, discount-price bookings should be accepted as long as their value exceeds that of anticipated/expected full-price bookings. Brumelle et al. [101] translated that rule as: Suppose that we have y units of capacity of computing resource remaining and we receive a request from discount-price customer (class 2). If we accept request from discount-price customer (class 2) we will have revenue p_2. If we reject request from discount-price customer (class 2) we will have revenue $p_1 * P(d_1 > y)$. $P(d_1 > y)$ is the probability that full-price demand is greater than y. So we accept request from discount-price customer (class 2) if $p_2 \geq p_1 * P(d_1 > y)$. If a continuous distribution $F_1(x)$ is used to model demand, then the optimal protection level y is

$$p_2 = p_1 * P(d_1 > y)$$
$$p_2 = p_1 * [1 - P(d_1 \leq y)]$$
$$p_2 = p_1 - p_1 * P(d_1 \leq y)$$
$$p_1 * P(d_1 \leq y) = p_1 - p_2$$
$$P(d_1 \leq y) = \frac{p_1 - p_2}{p_1}$$
$$y = {F_1}^{-1}\left(\frac{p_1 - p_2}{p_1}\right)$$
$$y = \mu + \delta\, \Phi^{-1}\left(\frac{p_1 - p_2}{p_1}\right) \quad (3)$$

Where ${F_1}^{-1}$ refers to the inverse cumulative distribution function of full-price customer/demand, μ refers to mean of full-price customer/demand, δ refers to standard deviation of full-price customer/demand, and Φ^{-1} refers to the inverse standard normal cumulative distribution function. At the point where $y = \mu + \delta\, \Phi^{-1}\left(\frac{p_1 - p_2}{p_1}\right)$, the risk of spoilage and dilution is exactly balanced. We know that we cannot sell our computing resource more than the computing capacity C, in this case protection level for full-price customers is

$$y = \min\left[\mu + \delta\, \Phi^{-1}\left(\frac{p_1 - p_2}{p_1}\right), C\right] \quad (4)$$

Booking limit for discount-price customer is $b = C - y$, which means that after b computing resources has been reserved by discount-price customers, the discount-price is closed. If the next incoming reservation request is discount-price customer, reject this incoming reservation.

5.2.2.2 n Classes Capacity Allocation Problem (Heuristics approach)

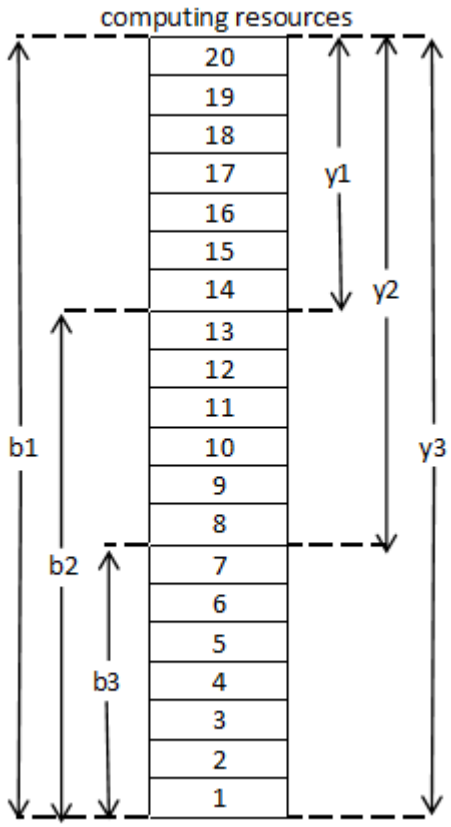

Figure 5.3: The relationship between booking limit and protection level

Airline industries are prolific price differentiators, often offering tens classes on a flight [46], but in grid computing, we can have two or three class capacity allocation problem [95]. In three class capacity allocation problem we have class 1, called as premium user, class 2, called as business user, and class 3, called as economy user. Booking limit for this 3 class user is always $b_3 \leq b_2 \leq b_1$, where b_3 is a booking limit for class 3 customers, and so on. b_1 is always the same as capacity of resource computing C. Protection level for 3 class user is always $y_1 \leq y_2 \leq y_3$. y_3 is always the

same as capacity of resource computing C. The relation between these booking limit and protection level is depicted in Figure 5.3.

From Figure 5.3 we have 20 computing resources or equivalently $C = 20$ and we can derive 4 equations which are:

$$C = b_1$$
$$C = b_2 + y_1$$
$$C = b_3 + y_2$$
$$C = y_3$$

For n class capacity allocation problem we have:

$$C = b_1$$
$$C = b_2 + y_1$$
$$C = b_i + y_{(i-1)} \quad (5)$$
$$C = y_n \quad (6)$$

One can compute booking limit for n class capacity allocation problem with a heuristic approach with three reasons [102]. First is that simply a case of practice being one step ahead of the underlying theory. Second, heuristics are widely used because they are simpler to code, quicker to run, and generate revenues that in many cases are close to optimal. Third, reflecting the philosophy that it is better to be "approximately right" than it is to be "precisely wrong."

Belobaba [103] [104] has introduced heuristic approach to calculate booking limit for n class capacity allocation problem, namely Expected Marginal Seat Revenue (EMSR). EMSR comes in two flavors: EMSR-a and EMSR-b. Both are based on approximating n class capacity allocation problem by a series of two class capacity allocation problems and applying Littlewoods rule to get the solution.

EMSR-a (Expected Marginal Seat Revenue version a) is based on the idea of calculating protection levels for each of the higher classes relative to the current class using Littlewood's rule. The protection level of higher class is the summation of all

protection levels of each higher class relative to the current class. For example, if we are in class $k+1$, EMSR-a would calculate a protection level for class k relative to class $k+1$ (denoted by ${y_k}^{k+1}$) using Littlewood's rule, then calculate a protection level for class $k-1$ relative to class $k+1$ (denoted by ${y_{k-1}}^{k+1}$) using Littlewood's rule, and so on, and finally calculate protection level for class 1 relative to class $k+1$ (denoted by ${y_1}^{k+1}$) using Littlewood's rule. Protection level of class k relative to class $k+1$ is calculated as follows:

$$ {y_k}^{k+1} = {F_k}^{-1}\left(\frac{p_k - p_{k+1}}{p_k}\right) $$

$$ {y_k}^{k+1} = \mu_k + \delta_k \, \Phi^{-1}\left(\frac{p_k - p_{k+1}}{p_k}\right) \tag{7} $$

where ${F_k}^{-1}$ refers to the inverse normal cumulative distribution function of class k demand, μ_k refers to mean of class k demand, δ_k refers to standard deviation of class k demand, and Φ^{-1} refers to inverse standard normal cumulative distribution function. Then protection level of class k is the summation of protection level of class k and higher relative to class $k+1$ as shown below

$$ y_k = \sum_{j=1}^{k} {y_j}^{k+1} \tag{8} $$

Refer to equation 5 and 6, booking limit for class $k+1$ is

$$ b_{k+1} = y_{k+1} - y_k \tag{9} $$

where y_{k+1} is the capacity C if $k+1$ is the lowest class.

EMSR-b (Expected Marginal Seat Revenue version b) assumes that a customer displaced by an additional booking would be paying a price equal to a weighted average of future price. EMSR-b creates an artificial class with demand equals to the sum of the demands for all the future periods, price equals to the average expected price from future booking and standard deviation equals to the square root of the

summation of standard deviation square of all future booking. Then use Littlewood's rule to calculate the booking limit of the current class k with respect to the artificial class.

Assume that demands in all classes follow normal distribution. One can calculate booking limit of class $k+1$ that has a price p_{k+1} by creating a virtual class with mean demand μ, price p, and standard deviation δ given by

$$\mu = \sum_{x=1}^{k} \mu_x \tag{10}$$

$$p = \sum_{x=1}^{k} p_x \mu_x / \mu \tag{11}$$

$$\delta = \sqrt{\sum_{x=1}^{k} {\delta_x}^2} \tag{12}$$

One can use Littlewood's rule for normal distribution to calculate booking limit for class $k+1$ and protection level for artificial class is

$$y_k = \min\left[\mu + \delta\, \Phi^{-1}\left(1 - \frac{p_{k+1}}{p}\right),\ y_{k+1}\right] \tag{13}$$

where y_{k+1} is equal to C if $k+1$ is the lowest class. Booking limit for class $k+1$ can be calculated as

$$b_{k+1} = y_{k+1} - y_k \tag{14}$$

where y_{k+1} is equal to C if $k+1$ is the lowest class.

Belobaba [104] in their experiment compared the revenue performance between EMSR-a, EMSR-b and the optimal revenue. The result showed that EMSR-b was consistently within 0.5% of the optimal revenue, while EMSR-a in some cases fall into 1.5% of the optimal revenue. Tallury et al. [105] ran a series of simulations comparing the performance of EMSR-b, EMSR-a and optimal revenue flight with

modified instance of the data reported by Wolmer [106]. Their result of the simulation showed that performance of EMSR-b is better than EMSR-a, however both strategies performed within 1% of the optimal revenue.

5.3 Revenue Management System for Grid System

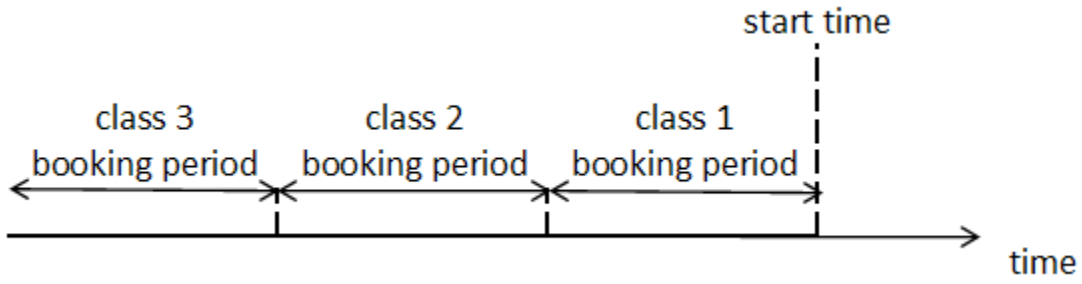

Figure 5.4: Booking periods for class 3, class2 and class 1 users

Sulistio et al. [107] [108] has incorporated the revenue management system in grid system. They use EMSR-a to calculate the booking limit and they also implemented the over booking. In our approach we use EMSR-b to calculate the booking limit because the performance of EMSR-b is better than EMSR-a [105]. We don't use over booking because we believe that it is impossible to implement it. We can use revenue management system in grid computing for n classes of users, but in our experiment we use three classes of users. In this case we will have three class users, i.e. class 1 is premium user with price p_1, class 2 is business user with price p_2 and class 3 is budget user with price p_3. Figure 5.4 shows the booking period for class 3 users, class 2 users and class 1 users. According to market segmentation and price discrimination, administrator sets $p_1 > p_2 > p_3$. Initial booking limit is initialized as; b_3 is booking limit for user class 3 users, b_2 is booking limit for a user class 2 users and b_1 is booking limit for class 1 users as depicted in Figure 5.5. Capacity of computing resource is also initialized as $MaxCN$. EMSR-b [104] is used to update initial booking limit in a particular time intervals.

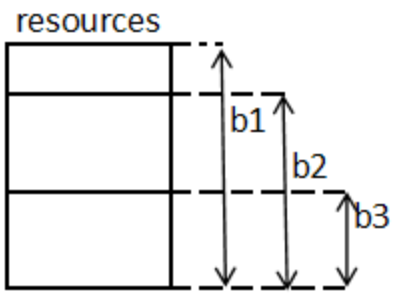

Figure 5.5: Booking limits for class 3, class2 and class 1 users

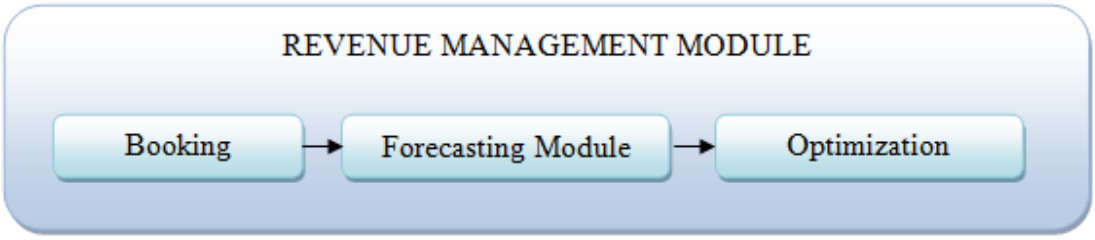

Figure 5.6: Revenue Management Module

Components inside Revenue Management Module are depicted in Figure 5.6. These components are, booking module, forecasting module, and optimization module.

1. Booking Module.

   Upon receiving reservation from Planning Module, Booking Module determines the class of the incoming reservation according to its booking period. If the reservation comes from class 3 user, then booking module decides whether this reservation is accepted or rejected according to its booking limit. If booking limit of its respective class is less than compute node needed, then "REJECTED" status for this reservation is communicated to Planning Module. We reject this reservation to accommodate future booking for higher class which in turn generates increase in revenue. If booking limit of its respective class is greater or equal to compute nodes needed by the reservation then status "CONFIRMED" is communicated to Planning Module, and initial booking limits of its respective class (b_3) and all higher class (b_2 and b_1) are updated, because of nesting booking

limit nature shown in Figure 5.5. Revenue Management Module upon confirming a reservation calculates the price, and updates revenue collected.

2. Forecasting Module.

   Upon sending status "CONFIRMED" to Planning Module, Booking module send the class, compute nodes needed, start time and finish time of the reservation to Forecasting Module. Upon sending status "REJECTED" to Planning Module, Booking module send the class, compute node needed, earliest start time and execution time of the reservation to Forecasting Module. Forecasting Module records those data to generate forecasts for all user class. The forecasts generated by Forecasting Module are probabilistic, that is, it predicts both a mean and a standard deviation for future demand

3. Optimization Module.

   The probabilistic forecasts are the input used by the "Optimization" module to update the initial booking limits. Initial booking limits are calculated by estimating the economic trade-off between accepting more discount bookings now versus having more capacity available to serve future bookings. We use EMSR-b [104] to calculate the initial booking limits for a given forecast.

Algorithm for updating initial booking limit for three class capacity allocations using EMSR-b can be seen below. Means and standard deviations of the first two classes as well as prices of all the three classes are given as input to the algorithm. Booking limits on reservations for the three classes are obtained as output for the algorithm.

Algorithm of updating booking limit is depicted in Algorithm 5.1 and explained as follows: Let us assume that y_1 is a protection level for class 1, y_2 is protection level of class 2, and y_3 is protection level for class 3. $vClass$ is a virtual class and c is capacity. Declaration of those variables is done in lines 12-21.

First we set the protection level of y_3 to capacity of class 3 (line 25). Mean, standard deviation, and price of virtual class ($vClass$) are calculated in the lines 26-28. Littlewood's rule is applied on class 3 and $vClass$ to calculate y_2 (lines 29 – 33). $inversScdf(x)$ is a function that gives inverse standard normal cumulative distribution function of given input x. To calculate y_1, we set members of virtual Class ($vClass$) to mean, standard deviation and price of class 1 (lines 35-37). y_1 is similarly calculated using Littlewood's rule on class 2 and $vClass$ with the corresponding capacity (lines 38-42). In the last step, b_1, b_2 and b_3 are calculated (liness 44-46).

Algorithm 5.1: Update Booking Limit

```
1  procedure UpdateBL(Input:m1, m2, s1, s2 : real, p1, p2, p3, maxCN :integer
2                     Output: b1, b2; b3 : integer)
3  /*Updating the booking limit of 3 class capacity allocation, with inputs
4  are: m1 is mean of class 1, m2 is mean of class 2,
5  s1 is standard deviation of class 1, s2 is standard
6  deviation of class 2, p1 is price of class 1, p2 is
7  price of class 2, p3 is price of class 3. Outputs
8  are: b1 is booking limit for class 1, b2 is booking limit for
9  class 2, b3 is booking limit for class 3*/
10
11 Dictionary :
12 y1 : integer //protection level for class one reservation
13 y2 : integer //protection level for class two reservation
14 y3 : integer //protection level for class three reservation
15 type BookingClass : <m : real //mean
16                      s : real //standard deviation
17                      p : integer //price
18                     >
19 vClass : BookingClass //virtual class
20 c : integer           //capacity
21 temp: integer
22
23 Algorithm :
24 c ← maxCN
25 y3 ← c
26 vClass.m ← m1+m2;
27 vClass.s ← sqrt(s1*s1+s2*s2)
28 vClass.p ← (m1*p1+m2*p2)/(m1+m2)
29 temp ← (integer) vClass.m+vClass.s*inversScdf(1-(p3/vClass.p))
30 if (temp>y3) then
31    y2 ← y3;
32 else
33    y2 ← temp
34
35 vClass.m ← m1
36 vClass.s ← s1
37 vClass.p ← p1
38 temp ← (integer) vClass.m+vClass.s*inversScdf(1-(p2/vClass.p))
39 if (temp>y2) then
40    y1 ← y2
41 else
42    y1 ← temp
43
```

```
44 b1 ← c
45 b2 ← y2 - y1
46 b3 ← y3 - y2
```

5.4 Example

To explain how to update initial booking limit works we introduce an example. We have $maxCN$ compute node (logical view). Let $maxCN$ is equal to 40 as logical nodes and we also have three classes user. Price and initial booking limit is depicted in Table 5.1. For example the given initial booking limit and the price for class 1 users in Table 5.1 implies the following "Class 1 users has initial booking limit 40 compute nodes and the price for them is Rs. 100,-".

Table 5.1: Initial Booking Limit and Price for three classes user

Users	Initial booking limit (b)	Price (Rs)
Class 1	40	100
Class 2	35	60
Class 3	30	40

Table 5.2: Mean and standar deviation

Users	Mean (μ)	Standard Deviation (δ)
Class 1	10	1.5
Class 2	13	1.7
Class 3	20	2.3

Mean and standard deviation of class 1 and class two users is depicted in Table 5.2. For example the given initial booking limit and the price for class 1 users in Table

5.2 implies the following “For Class 1 users mean and standard deviation are 10 and 1.5 respectively”

Updating the initial booking limit can be done as follows:

1. In order to update the initial booking limit for class 3 users we have to create virtual class for higher classes i.e. $vClass$ with mean (μ), standard deviation (δ) and price (p) as follows (derived from equation 10, 11 and 12):

$$\mu = \sum_{x=1}^{2} \mu_x = 10 + 13 = 23$$

$$p = \sum_{x=1}^{2} p_x \mu_x / \mu = \frac{100 * 10}{23} + \frac{60 * 13}{23} = 77.39130435$$

$$\delta = \sqrt{\sum_{x=1}^{2} {\delta_x}^2} = \sqrt{1.5^2 + 1.7^2} = 2.26715681$$

2. Apply Littlewood's rule to class 3 and $vClass$ to obtain protection level for $vClass$ (y_2, derived from equation 4).

$$y_2 = \min\left[\mu + \delta\, \Phi^{-1}\left(\frac{p - p_2}{p}\right), C\right]$$

$$y_2 = \min\left[\left[23 + 2.26715681\, \Phi^{-1}\left(\frac{77.39130435 - 40}{77.39130435}\right)\right], 40\right]$$

$$y_2 = \min[22\,, 40]$$

$$y_2 = 22$$

3. Apply Littlewood's rule to class 2 and class 1 and the capacity is equal to $y_2 = 22$ (since y is protection level for class 1 and 2).

$$y_1 = \min\left[\mu_1 + \delta_1\, \Phi^{-1}\left(\frac{p_1 - p_2}{p_1}\right), C\right]$$

$$y_1 = \min\left[\left[10 + 1.5\, \Phi^{-1}\left(\frac{100 - 60}{100}\right)\right], 22\right]$$

$$y_1 = \min[9, 22]$$

$$y_1 = 9$$

4. Calculate initial booking limit as for class 3 user (b_3) follows:

$$b_3 = C - y_2 = 40 - 22 = 18$$

5. Calculate initial booking limit for class 2 user (b_2) as follows:

$$b_2 = C - y_1 = 40 - 9 = 31$$

6. b_1 and y_3 are all the same as capacity of the compute node i.e $b_1 = 40$ and $y_3 = 40$.

After updating the initial booking limit we have a new booking limit for each class user as depicted in Table 5.3.

Table 5.3: Booking limit before and after update

	Before update	**After update**
$\boldsymbol{b_1}$	40	40
$\boldsymbol{b_2}$	35	31
$\boldsymbol{b_3}$	30	18

5.5 Discussion

In this chapter we have discussed about revenue management system. Revenue management strategy can be done by doing market segmentation and price

discrimination. According to market segmentation and price discrimination we can calculate capacity allocation for each class user. In this chapter we also discussed the detail components inside the Revenue Management Model. We also have presented the algorithm to update the initial booking limit using EMSR-b method.

6 A New Grid System Architecture: Experiments and Results

This chapter proposes architecture of our work as a complete system. It shows the interaction between different components in the model to perform the proposed advance reservation scheduling strategy (FCFS-EDS) by incorporating revenue management. To demonstrate the efficacy of the proposed FCFS-EDS strategy, the data structure, revenue management etc. a novel simulator has been designed and implemented. This simulator has a potential to accommodate reservation for both Parametric as well as MPI job types. The software needed for this simulator is Eclipse SDK 3.5.0 Galileo, with java programming language and the hardware needed is laptop with processor Intel Pentium M Processor 1.73 GHz with 512MB RAM and 80 GB Hard disk drive. The chapter concludes with experimental results.

6.1 A Proposed Architecture

Our proposed architecture as a complete system, depicted in Figure 6.1, shows interaction between relevant components in the model. Components which are involved in this model are: Administrator, User, Planning and Reservation Module, Resource Allocation, and Revenue Model.

a. Administrator sets the initial conditions for the model, and prepares reports.
b. User submits the job and its description.

c. Planning Module schedules the incoming job from the user using FCFS-EDS scheduling strategy.

d. Revenue Management Module decides whether request from a user is to be accepted or rejected even if there are computing nodes (resources) available for the job. This requirement stems from anticipated future booking for higher class of jobs which could pay more.

e. Reservation Module maps the user job to required physical resource nodes, using our proposed FCFS-EDS's lemma.

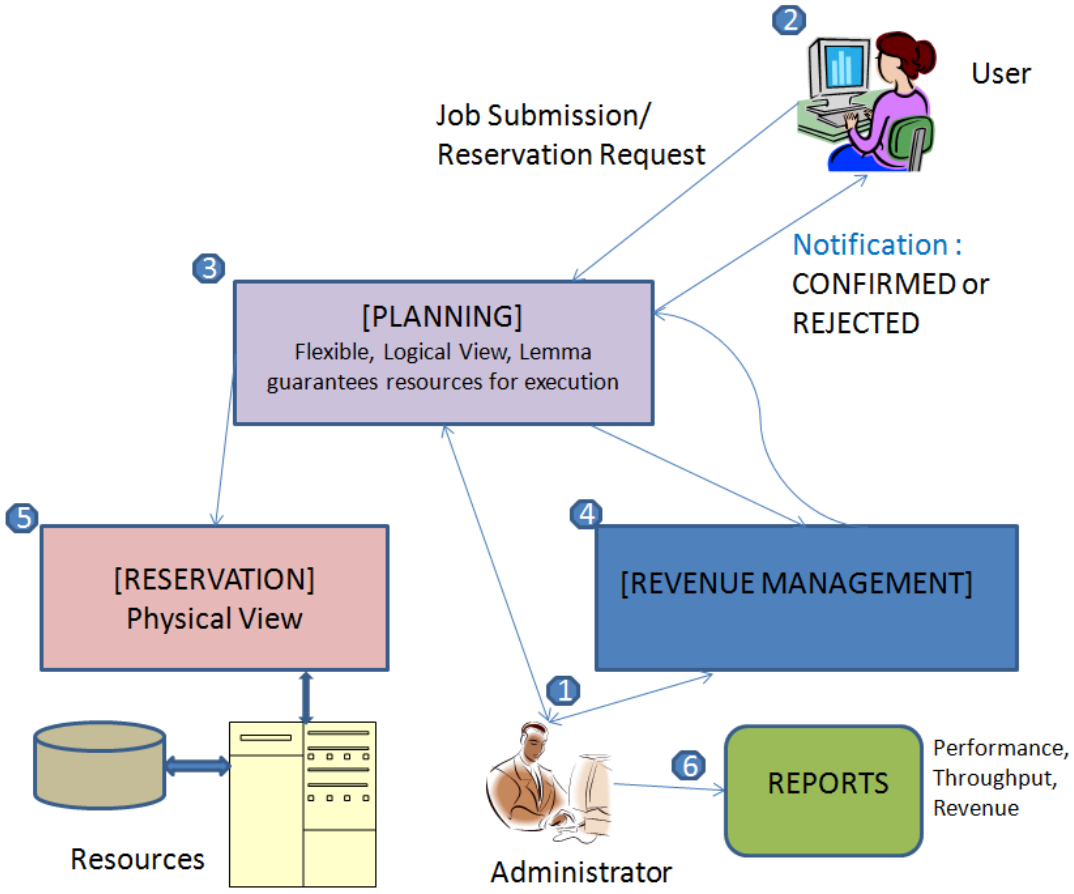

Figure 6.1: An architecture for Advance Planning, Reservation and Revenue Management System

The explanation of interaction between these components is now explained below:

STEP-1 (Initialization)

Administrator does initialization for "Planning" and "Revenue Management" Modules. In "Planning" Module, Administrator specifies the scheduling strategy (FCFS-EDS) and defines parameters for the job submission like: earliest start time, latest start time, duration of the job, $userId$ and $jobId$, number of compute nodes needed and capacity of computing resources. In "Revenue Management" Module the Administrator initializes the number of user classes and initializes capacity allocations (initial booking limit) for those users. It also initializes prices and method used for updating initial booking limits for each user class. Periodically generates reports as needed based on initialization.

STEP-2 (Job Submission Interface)

User defines parameters for Job Submission. The parameters are: $userId$, $jobId$, earliest start time, latest start time, duration of the job, and number of compute nodes needed. User sends the job profile to "Planning" Module.

STEP-3 (Planning)

In the "Planning" Module, jobs received are processed to see if jobs can be accepted based on the description (parameters). If accepted, submitted job details are sent to "Revenue Management" Module else user is notified "REJECTED" status for the job because adequate resources (compute nodes) are not available. The job is also deleted from the list. On receiving status from "Revenue Management" Module do the following actions:

- If "CONFIRM" status received from STEP-4 then "CONFIRMED" notification is sent to user.
- If "REJECT" status received from STEP-4 then "REJECTED" notification is sent to user. The job is also deleted from the list.

STEP-4 (Revenue Optimizing)

"Revenue Management" Module upon receiving job profile from STEP-3sets the class of user job (i.e. class j user), if booking limit of its respective class is less than compute node needed, then "REJECT" status for a job is communicated to STEP-3. "Revenue Management" Module rejects this job to accommodate future booking for higher class which in turn generate increase revenue. If booking limit of its respective class is greater or equal to compute node needed by the job then status "CONFIRM" is communicated to STEP-3, and booking limits are updated. "Revenue Management" Module upon confirming a job calculates the price, and updates revenue collected.

STEP-5 (Binding Process)

Here in "Reservation" Module, confirmed job in STEP-3 are scheduled on required physical resources. According to the proposed FCFS-EDS's lemma, "Reservation" Module will always find resources for jobs which are scheduled in logical view, and then "Reservation" Module executes the job on the actual compute nodes.

STEP-6 (Report Generation)

Report Module generates Revenue Report, as required/defined by the Administrator. Example: window of size one hour.

6.2 Work Load Generator

In order to examine the performance of our proposed advance planning and reservation scheduling strategy, FCFS-EDS, we have developed a workload generator. The output of the workload generator is the input of our proposed advance planning and reservation scheduling FCFS-EDS. In order to generate the workload, we have to specify the characteristic of the workload. Characteristic of the workload that we have to specify are:

1. The rate of the incoming reservation in each time slot. This rate of incoming reservation follows a Poisson distribution.

2. The range of execution time (t_e) of each reservation. For example the range of execution time can be 3 time slots to 20 time slots. These execution times are uniformly distributed.

3. The range of book ahead or the earliest start time (t_{es}) of each reservation. For example the range of book ahead or the earliest start time can be 0 time slots to 24 time slots. These book aheads or earliest start times are uniformly distributed.

4. Percentage of the flexible advance reservation. These flexible advance reservations are selected randomly.

5. The range of relax time (t_r) for each flexible advance reservation. For example the relax time can be the number between 1 to 12 time slots. These relax times of flexible advance reservation is uniformly distributed.

6. The range of the number of compute node needed. For example the number of compute node needed is between 1 to 5 compute nodes. These numbers of compute node needed are uniformly distributed.

6.3 Experiments and Results

This chapter then presents following experiments that have been done.

- Initially we have done the experiments regarding our proposed advance planning and reservation scheduling strategy, FCFS-EDS. These experiments deal with two parts: parametric jobs and MPI Jobs. In each part, we compare the utilization ratio with the existing advance reservation strategy which is not using advance planning and reservation.
- Next, we have done the experiments for the proposed revenue management which is incorporated in our advance planning and reservation strategy, FCFS-EDS. We compare the collected revenue of our FCFS-EDS advance planning and reservation strategy with and without incorporating revenue management system. These experiments again comprise of two parts, i.e. FCFS-EDS,

advance planning and reservation strategy for parametric job and FCFS-EDS, advance planning and reservation strategy for MPI job.

6.4 FCFS-EDS for parametric job

We have done eight experiments on our proposed advance planning and reservation scheduling strategy FCFS-EDS for parametric job. The workload or user requests (partly influenced by [107]) of these experiments have these characteristics:

a. The rate of incoming reservation requests are assumed to follow poison distribution with mean as depicted in Table 6.1.

b. Execution time (t_e) for reservation requests are between 5 to 48 timeslots distributed uniformly.

c. Earlier starting time (t_{es}) for reservation requests are between 0 to 48 timeslots, distributed uniformly.

d. Percentage of user request that are for flexible advance reservation, as depicted in Table 6.1 (selected randomly).

e. Relax time (t_r) for reservation requests are between 1 to 24 timeslots distributed uniformly and $t_{ls} = t_{es} + t_r$

f. In the experiment it is assumed that a time slot is equal to 5 minutes (clock time).

We compare the performance of the proposed method (FCFE-EDS with advance planning) and an existing approach (flexible advance reservation strategy without advance planning). With above inputs and total number of compute node is 30 ($maxCN = 30$), the utilization factors of both strategies are measured. For those eight experiments (Table 6.1), the comparisons of resource utilizations of both strategies are shown in Figure 6.2 to Figure 6.9. Percentage of utilization factor is calculated within sliding window of size 12 time slots (1 hour). Figure 6.2 to Figure 6.9 show that

FCFS-EDS yields better utilization than the traditional strategy (without advance planning).

Table 6.1: Experiments for parametric job

No	Rate	Percentage (%)
1	2	25
2	2	50
3	2	75
4	2	100
5	3	25
6	3	50
7	3	75
8	3	100

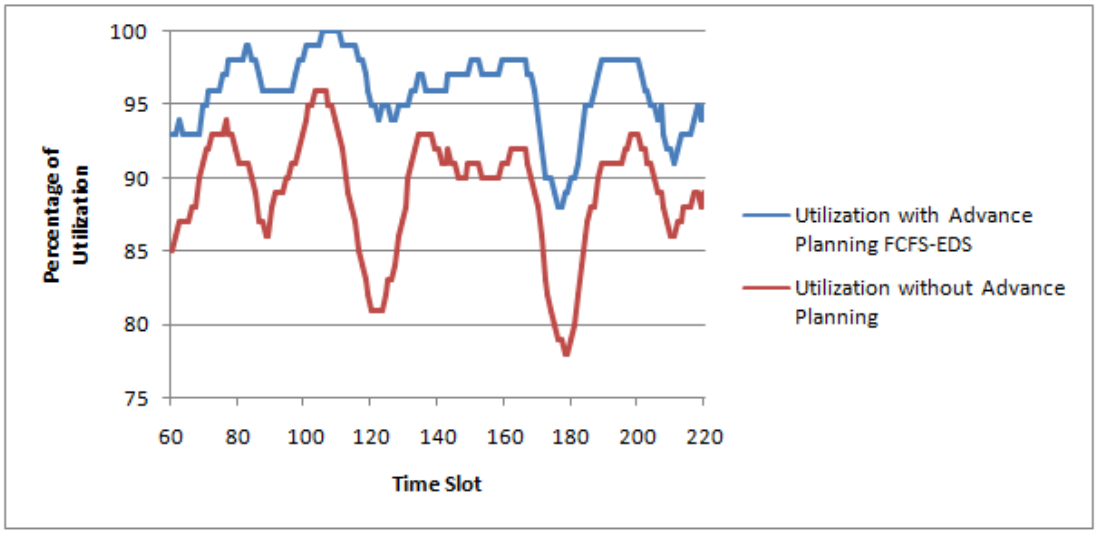

Figure 6.2: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #1)

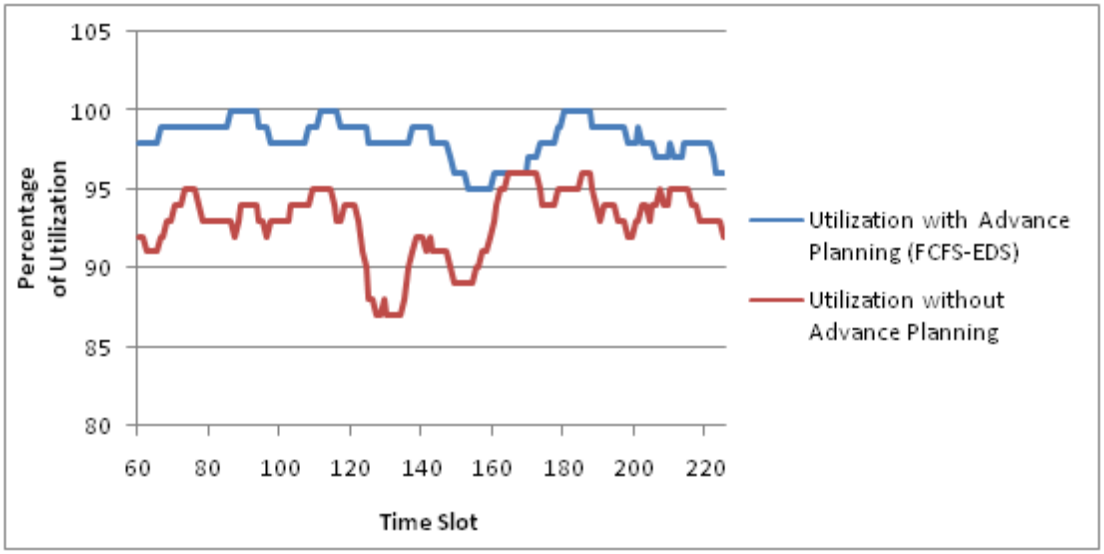

Figure 6.3: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #2)

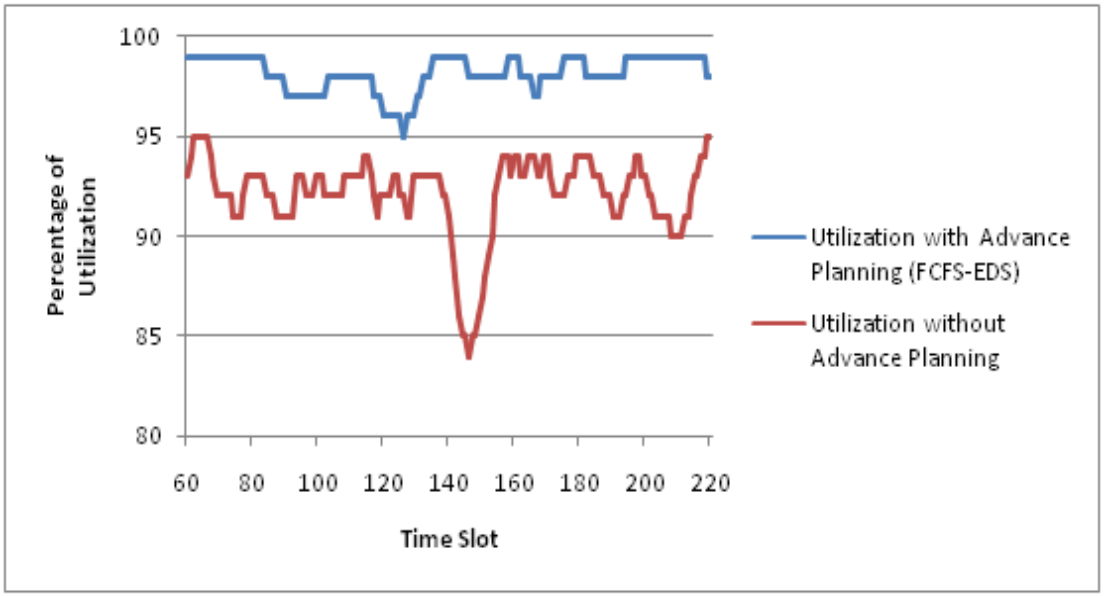

Figure 6.4: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #3)

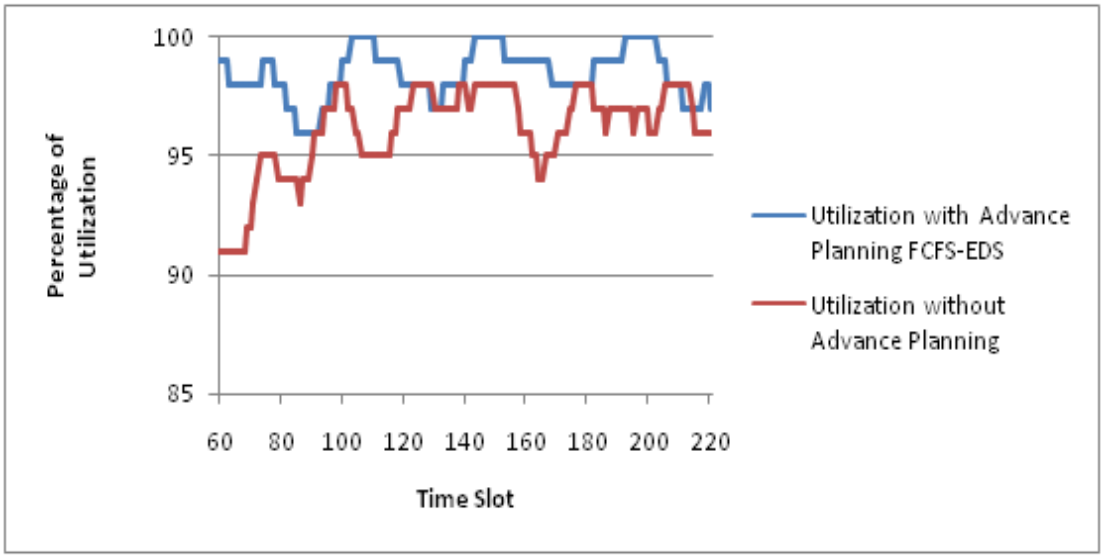

Figure 6.5: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #4)

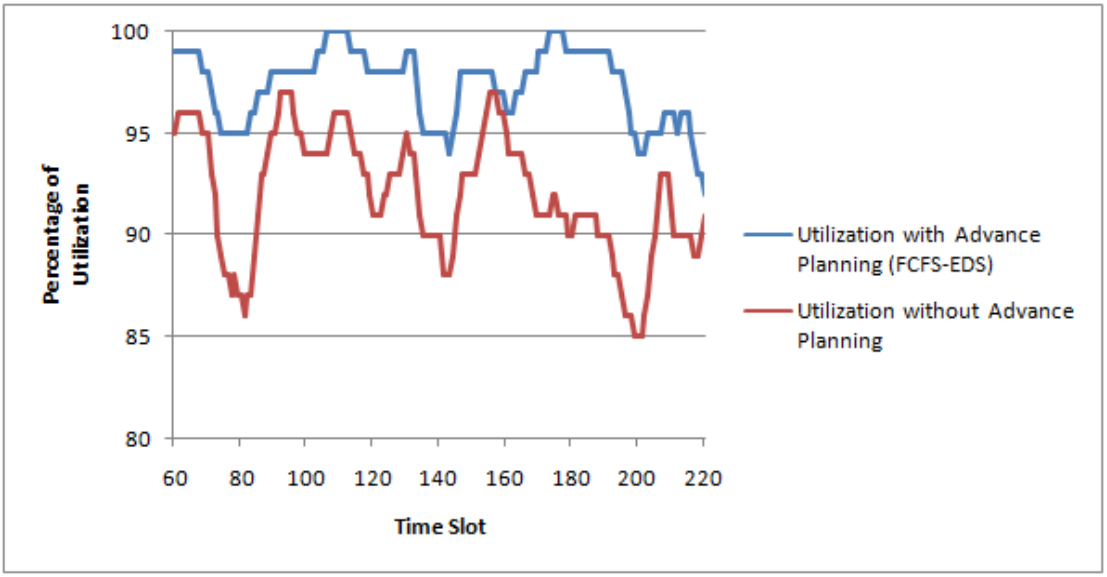

Figure 6.6: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #5)

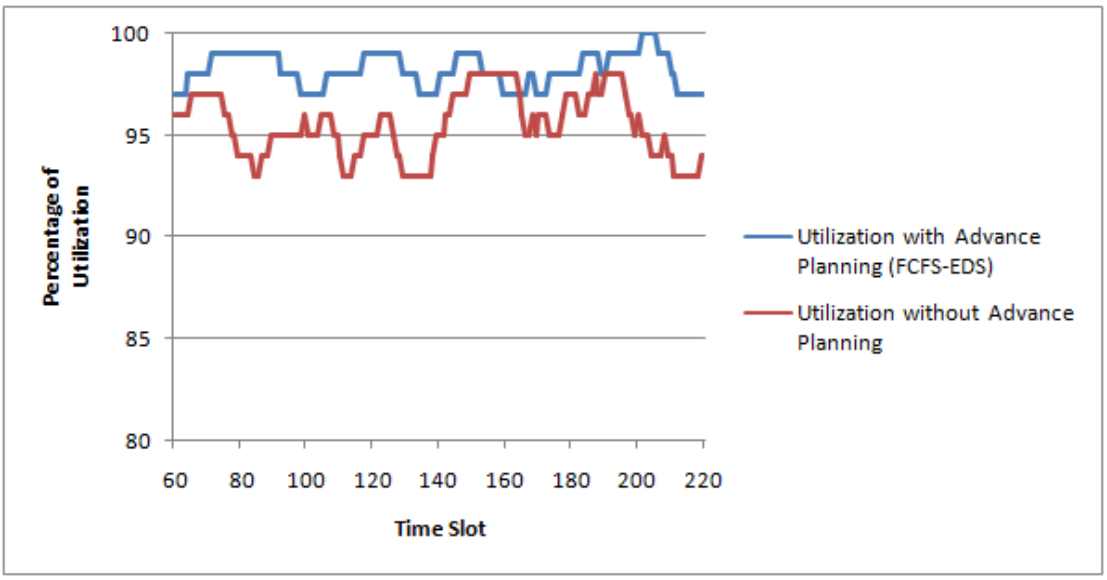

Figure 6.7: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #6)

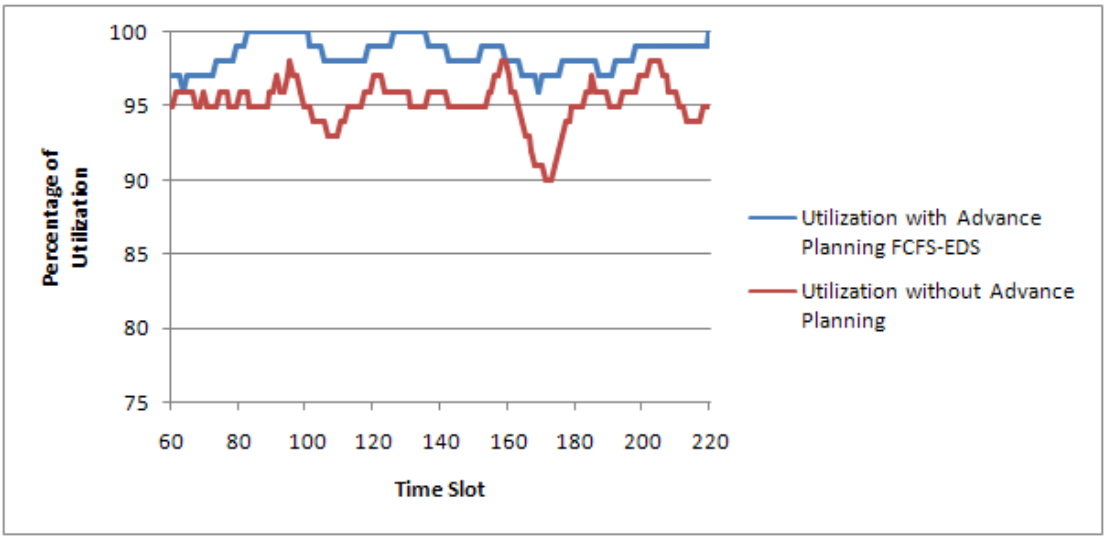

Figure 6.8: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #7)

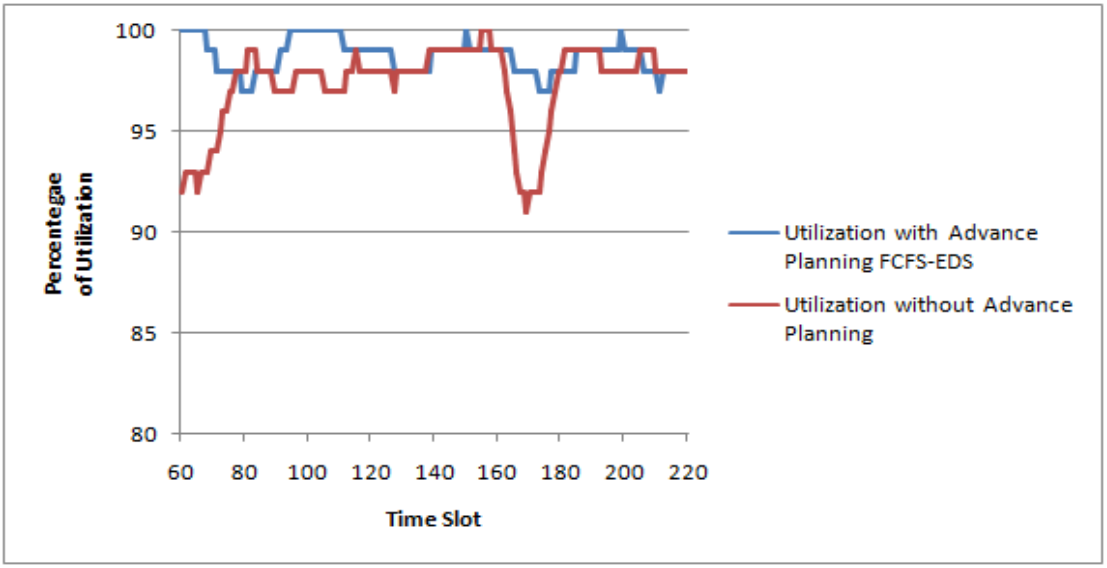

Figure 6.9: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for parametric job (experiment #8)

6.5 FCFS-EDS for MPI job

We have done the experiment for our proposed advance planning and reservation scheduling strategy FCFS-EDS for parametric job. The workload or user requests (partly influenced by [107]) of this experiment have these characteristics:

a. The rate of incoming reservation requests are assumed to follow poison distribution with mean as depicted in Table 6.2.
b. Execution time (t_e) for reservation requests are between 5 to 15 timeslots distributed uniformly.
c. Earliest starting time (t_{es}) for reservation requests are between 0 to 24 timeslots, distributed uniformly.
d. Percentage of user request that are for flexible advance reservation is depicted in Table 6.2 (selected randomly).
e. Relax time (t_r) for reservation requests are between 1 to 12 timeslots distributed uniformly and $t_{ls} = t_{es} + t_r$

f. Number of compute node needed ($numCN$) is between 1 to 5 compute nodes and distributed uniformly.

g. In the experiment it is assumed that a time slot is equal to 5 minutes (clock time).

Table 6.2: Experiments for MPI job

No	Rate	Percentage (%)
1	2	25
2	2	50
3	2	75
4	2	100
5	*3*	25
6	*3*	50
7	*3*	75
8	*3*	100

We compare the performance of our method (FCFS-EDS with logical view) and one with only physical view. With above inputs and total number of compute node is 30 ($maxCN = 30$), the utilization factors of both strategies are measured. For those eight experiments (Table 6.2), the comparisons of resource utilization of both strategies are shown in Figure 6.10 to Figure 6.17. Percentage of utilization factor is calculated within sliding window of size 12 time slots (1 hour). Figure 6.10 to Figure 6.17 show that FCFS-EDS yields better utilization than the traditional strategy (without advance planning).

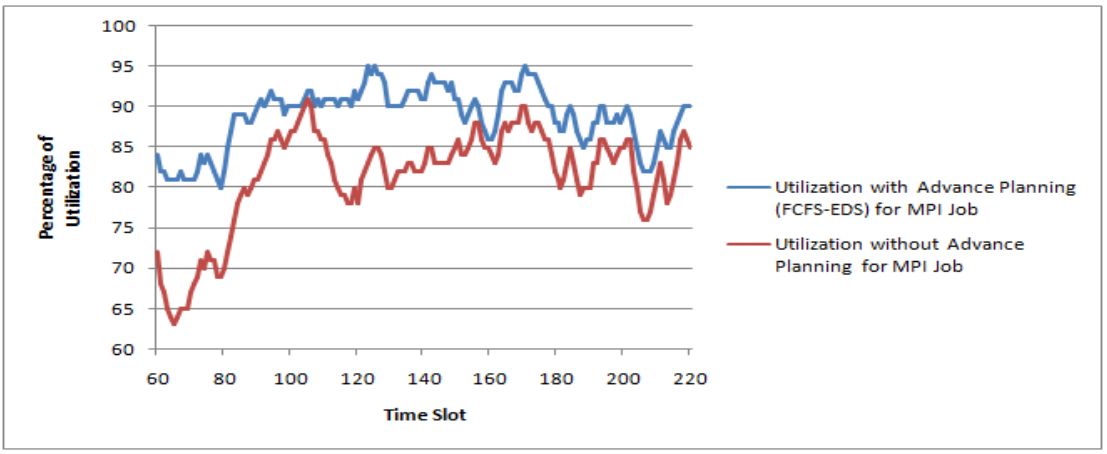

Figure 6.10: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #1)

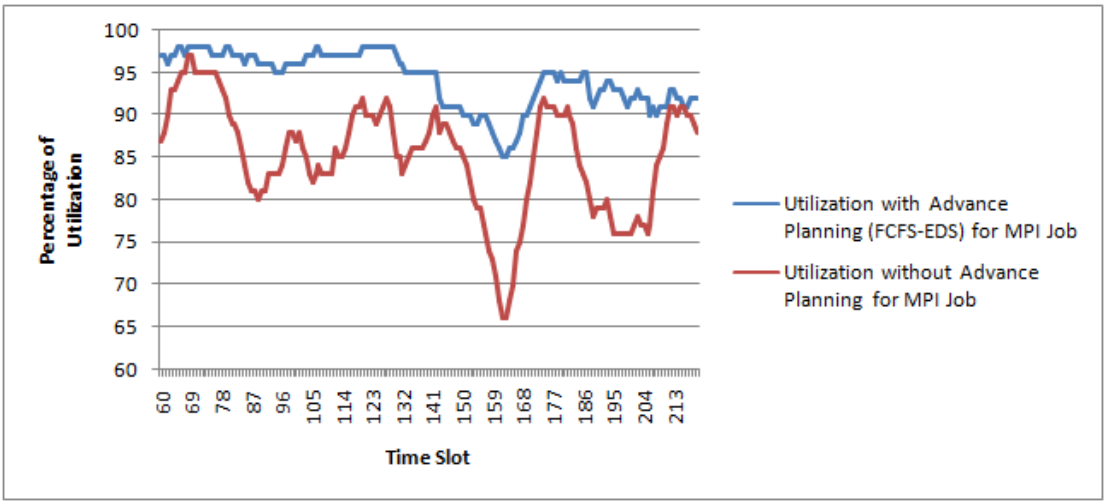

Figure 6.11: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #2)

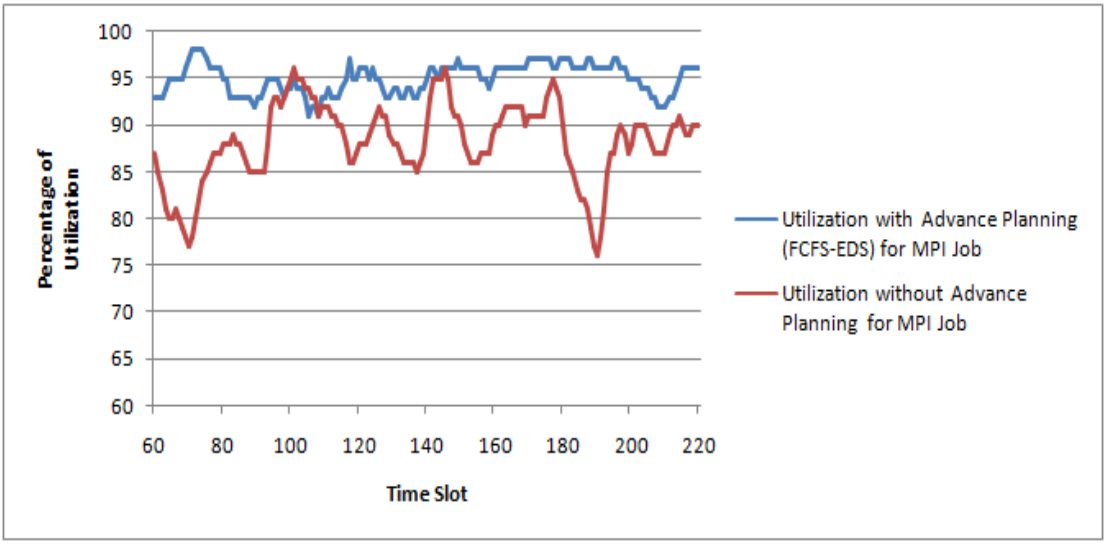

Figure 6.12: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #3)

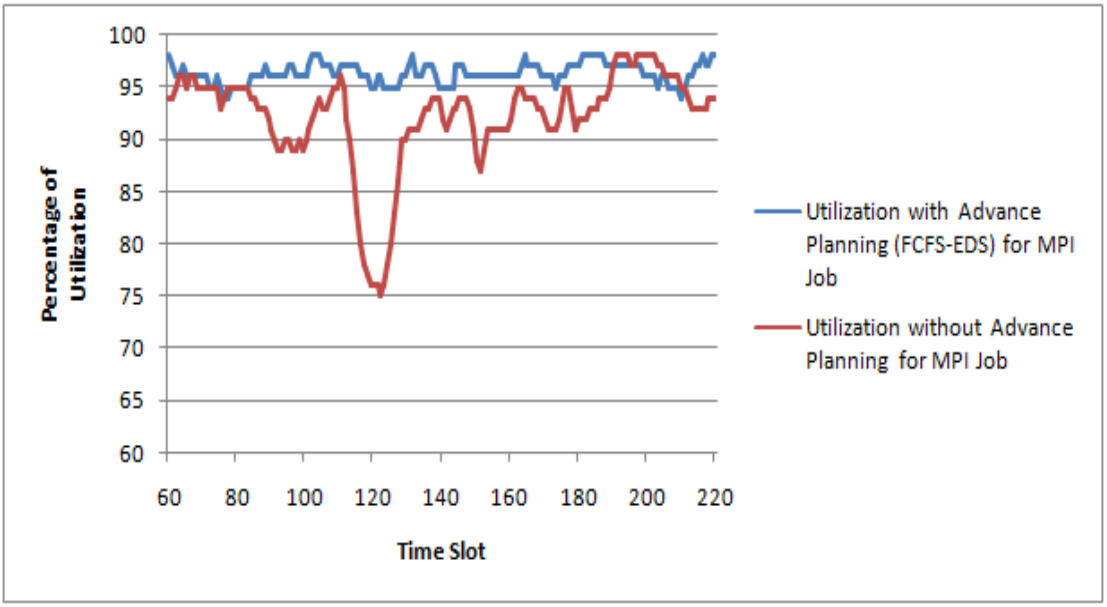

Figure 6.13: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #4)

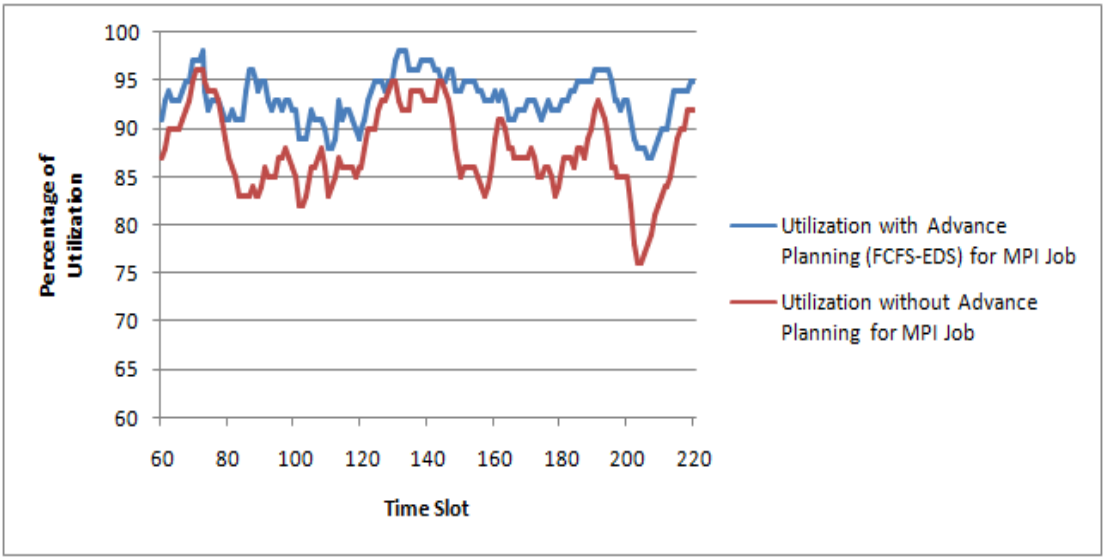

Figure 6.14: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #5)

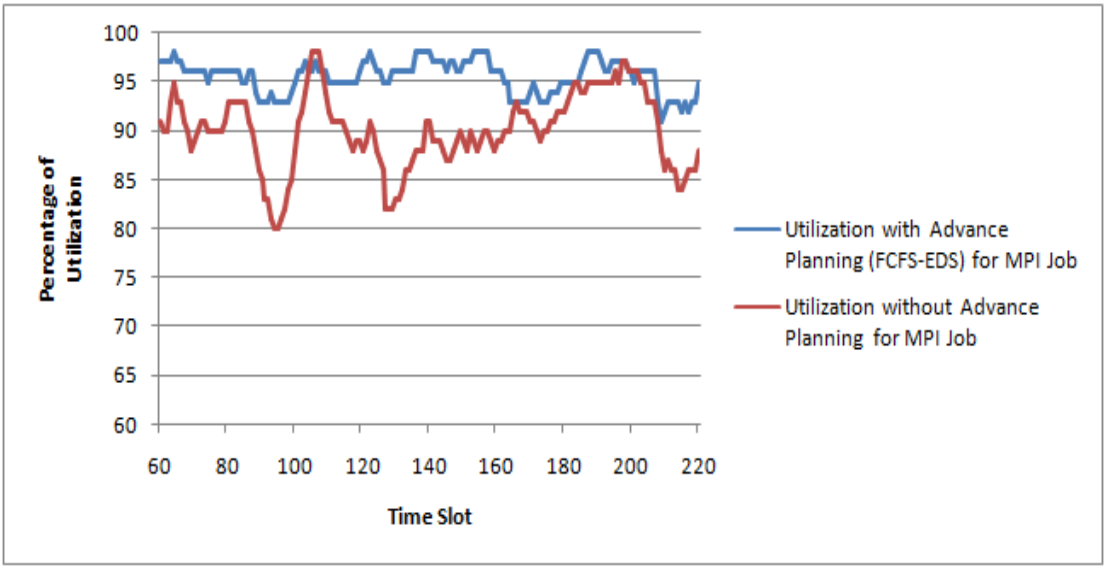

Figure 6.15: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #6)

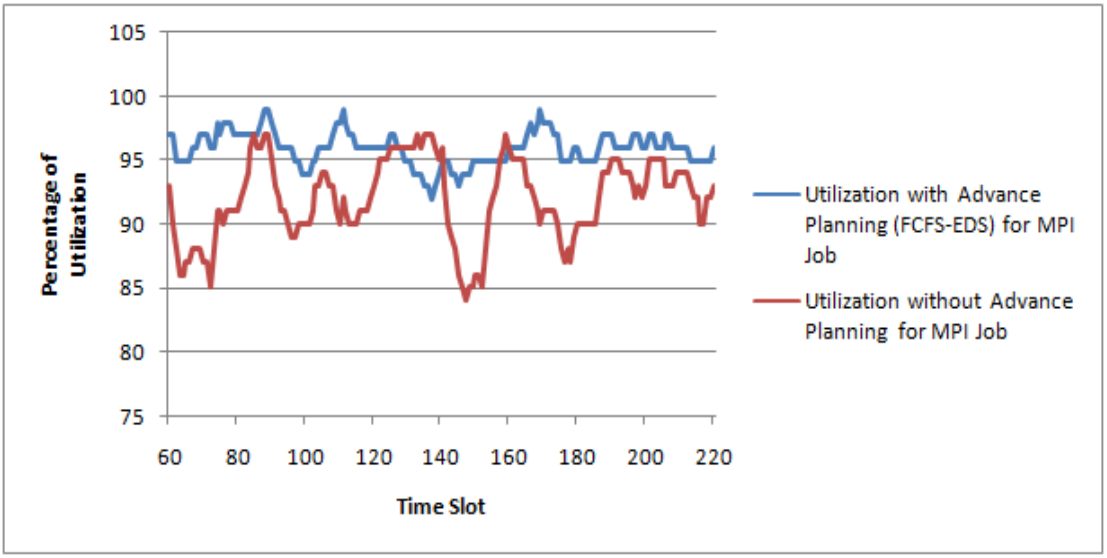

Figure 6.16: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #7)

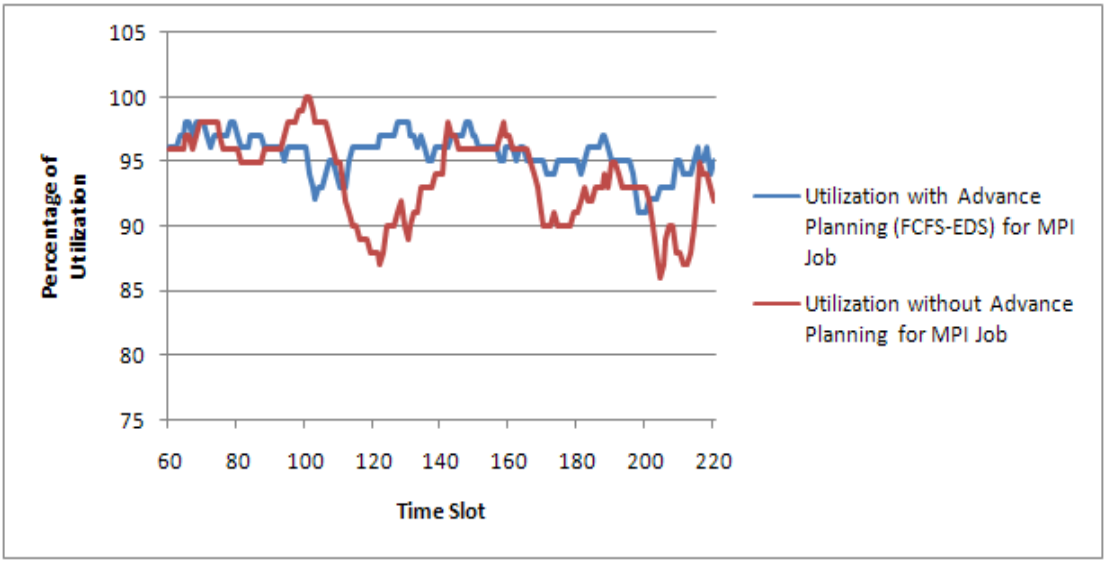

Figure 6.17: Comparison of percentage of utilization factor between scheduling with Advance Planning (FCFS – EDS) and flexible advance reservation without Advance Planning for MPI job. (Experiment #8)

6.6 Revenue Management System strategy for parametric job

In the experiment in incorporating revenue management for parametric job, first, we have to make an experimental setup. Experimental setups for incorporating revenue management, for parametric jobs are:

1. We have 3 class of user, which we call premium user, business user and economy user.
2. Premium users also called as class 1 users pay p_1, i.e. full price for compute node.
3. Business users also called as class 2 users pay p_2, i.e. discount price for compute node.
4. Economy users also called as class 3 users pay p_3, i.e. discount price for compute node.
5. We set p_1= Rs. 100,- i.e. full price for 1 timeslot on one compute node, p_2= Rs. 60,- i.e. full price for 1 timeslot on one compute node, and p_3= Rs. 40,- i.e. full price for 1 timeslot on one compute node.
6. Class 1 users reserve compute node between 0 to 12 timeslots in advance or book ahead is between 0 to 12 timeslots, class 1 users reserve compute node between 0 to 12 timeslots in advance, class 1 users reserve compute node between 0 to 12 timeslots in advance. These market segmentation and price discrimination can be seen in Table 6.1.

Table 6.3: Market Segmentation and Price Discrimination

Users	Book ahead	Price (Rs)
Class 1	$0 \leq ba \leq 12$	100
Class 2	$12 < ba \leq 24$	60
Class 3	$ba > 24$	40

7. Capacity of the compute node is 40 compute nodes.

8. Initial Booking Limit for class 1 users (b_1), class 2 users (b_2), and class 3 users (b_3) are depicted in Table 6.4. As depicted in Table 6.4, experiment #1 and #3 are designed to represent spoilage case, and experiment #2 and #4 are designed to represent dilution case.

We have done the experiments for our proposed advance planning and reservation scheduling strategy FCFS-EDS for parametric job. The workload or user requests of this experiment have these characteristics:

a. The rate of incoming reservation requests are assumed to follow poison distribution with mean as depicted in Table 6.4. If we use a very high rate of the incoming reservation then all computing nodes are reserved by class 1 users, which means there is no need to incorporate revenue management in it.
b. Execution time (t_e) for reservation requests are between 12 to 24 timeslots distributed uniformly.
c. Earlier starting time (t_{es}) for reservation requests are between 0 to 36 timeslots, distributed uniformly.
d. Percentage of user request that are for flexible advance reservation is assumed to at most 50% (selected randomly).
e. Relax time (t_r) for reservation requests are between 1 to 12 timeslots distributed uniformly and $t_{ls} = t_{es} + t_r$
f. In the experiment it is assumed that a time slot is equal to 5 minutes (clock time).

Table 6.4: Rate and Initial booking limit for the experiment

No	Rate	Initial b_1	Initial b_2	Initial b_3
1	3	40	10	5
2	3	40	30	35
3	4	40	10	5
4	4	40	30	35

In each experiment as depicted in Table 6.4, we have compared the revenue collected in sliding window of size 12 time slot (1 hour), between these three scenarios:

1. Revenue of FCFS-EDS scheduling strategy without implementing revenue management. In Figure 6.18 to Figure 6.21 they are shown as blue line.
2. Revenue of FCFS-EDS with implementing revenue management without updating initial booking limit. In Figure 6.18 to Figure 6.21 they are shown as shown in red line.
3. Revenue of FCFE-EDS with implementing revenue management with updating booking limit. In Figure 6.18 to Figure 6.21 they are shown as shown in green line. After updating the booking limit with EMSR-b formula, we have new initial booking limits as shown in Table 6.5 to Table 6.8.

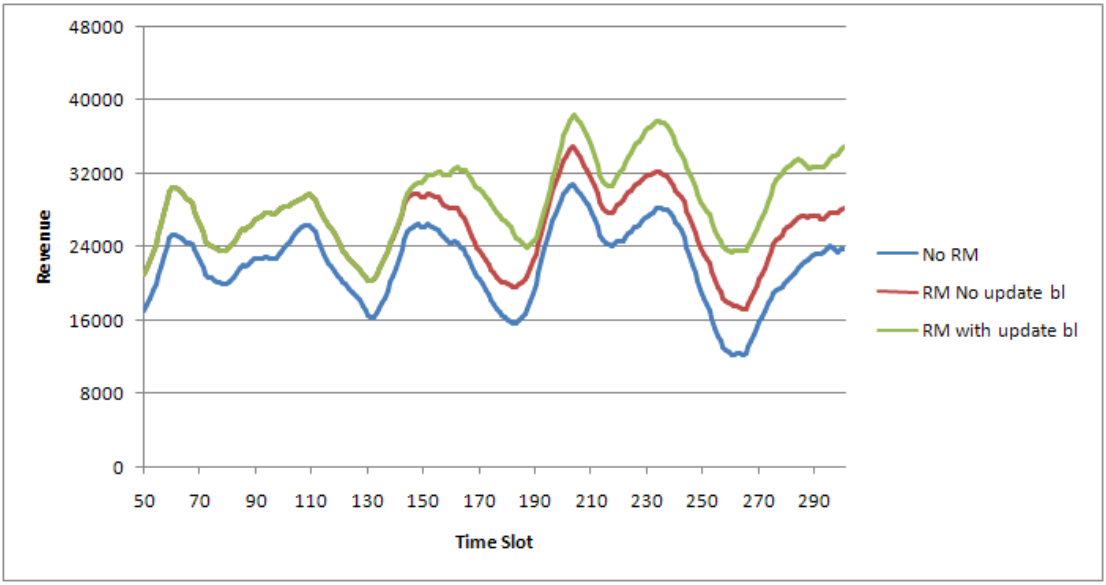

Figure 6.18: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #1)

Table 6.5: Value of initial booking limit before and after update, for parametric job (experiment #1)

	Before update	**After update**
$\boldsymbol{b_1}$	40	40
$\boldsymbol{b_2}$	10	22
$\boldsymbol{b_3}$	5	2

From Figure 6.18 (results of experiment #1), it can be seen that revenue of FCFS-EDS without revenue management has lower revenue than that with revenue management, as it gains additional revenue by accepting discount-price users. This revenue becomes much higher after updating the booking limit (depicted in Table 6.5). By accepting too less number of discount-price users, it hopes for future bookings that pay a full price of compute nodes. But then it may not see enough full-price booking to reserve compute nodes. This is called spoilage as the idle compute nodes become spoiled. To reduce the spoilage, it limits the reservation for class 3 users from 5 to 2 compute nodes, and for class 2 users from 10 to 22 compute nodes as depicted in Table. 6.5.

Table 6.6: Value of initial booking limit before and after update, for parametric job (experiment #2)

	Before update	**After update**
$\boldsymbol{b_1}$	40	40
$\boldsymbol{b_2}$	35	29
$\boldsymbol{b_3}$	30	5

From Figure 6.19 (results of experiment #2), it can be seen that revenue of FCFS-EDS without revenue management has lower revenue than that with revenue management, as it gains additional revenue by accepting discount-price users. This revenue becomes much higher after updating the booking limit (depicted in Table 6.6). By accepting too many discount-price users it rejects future booking of full-price users as resource capacity is limited, i.e. 40 compute nodes. This is called dilution since it

dilutes the revenue that could have received from saving additional compute nodes for full-price users. To anticipate future bookings that pay full price of compute nodes, it then limits the reservation from class 3 users from 30 to 5 compute nodes, and from class 2 users from 35 to 29 compute nodes as depicted in Table 6.6.

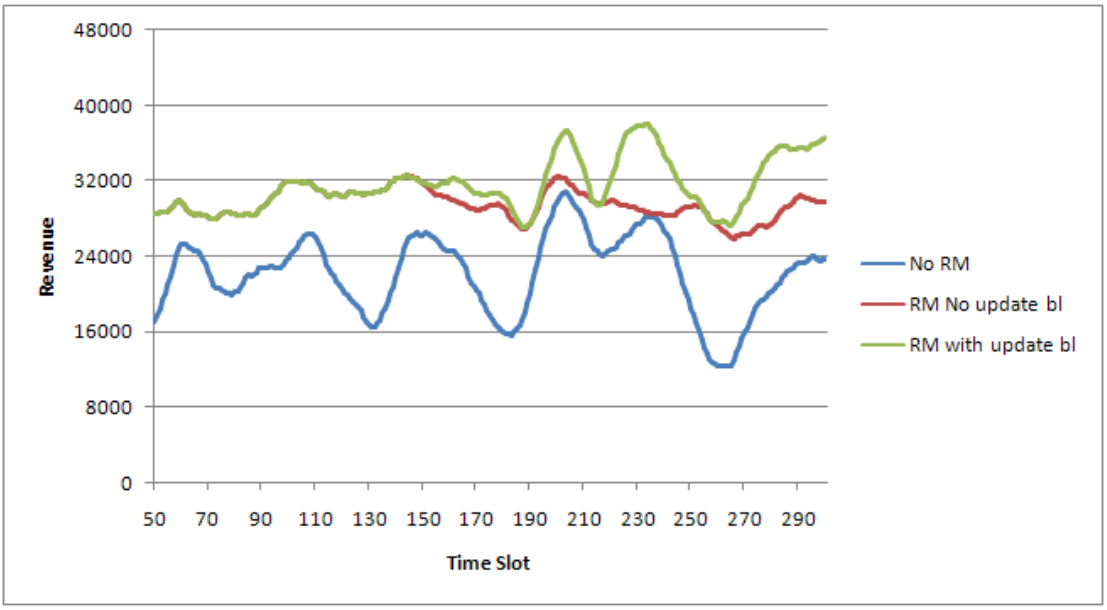

Figure 6.19: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking. (Experiment #2)

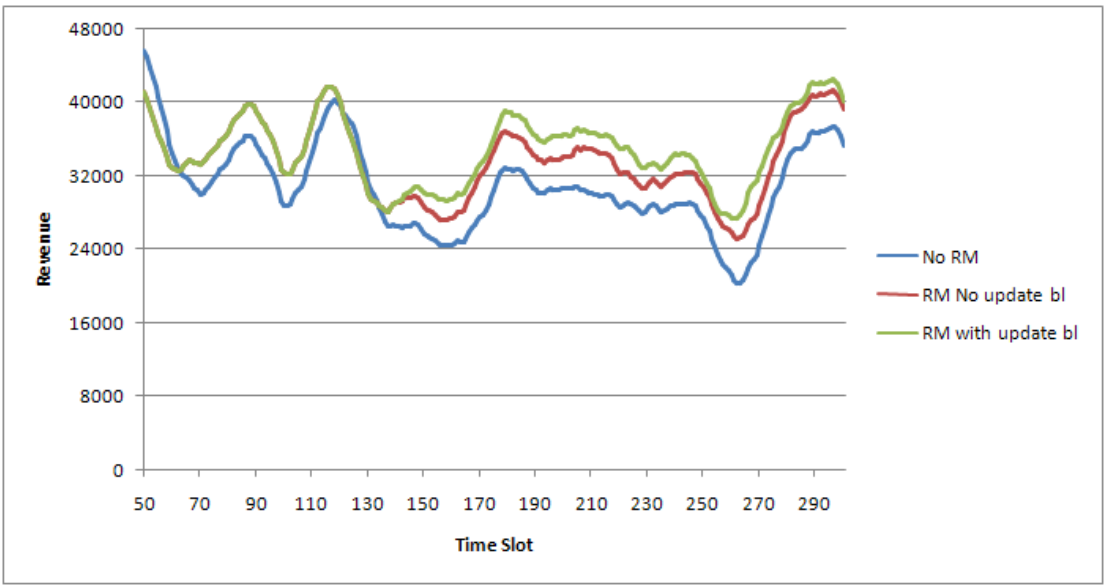

Figure 6.20: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking. (Experiment #3)

Table 6.7: Value of initial booking limit before and after update, for parametric job (experiment #3)

	Before update	**After update**
b_1	40	40
b_2	10	14
b_3	5	0

From Figure 6.20 (results of experiment #3), it can be seen that revenue of FCFS-EDS without implementing revenue management has the lower revenue than that with revenue management, as it gains additional revenue by accepting discount-price users. This revenue, even going higher, after updating the booking limit (depicted in Table 6.7). By accepting too less number of discount-price users, it hopes for future bookings that pay a full price of compute nodes. But then it may not see enough full-price booking to reserve compute nodes. This is called spoilage as the idle compute nodes become spoiled. To reduce the spoilage, it limits the reservation from class 3

users from 5 to 0 compute nodes, and from class 2 users from 10 to 14 compute nodes as depicted in Table. 6.7.

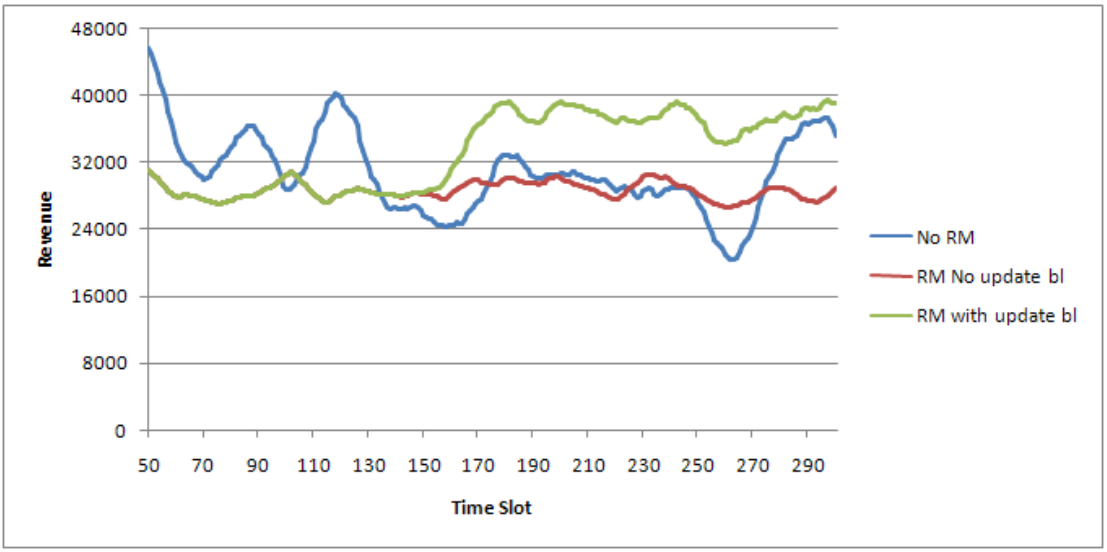

Figure 6.21: Comparison of revenue between FCFS-EDS scheduling strategy for parametric jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #4)

Table 6.8: Value of initial booking limit before and after update, for parametric job (experiment #2)

	Before update	**After update**
b_1	40	40
b_2	35	29
b_3	30	0

From Figure 6.21 (results of experiment #4), it can be seen that revenue of FCFS-EDS without revenue management has the higher revenue than that with revenue management. This is because it accepts too many discount-price users, which leads to rejecting full-price users. The revenue of FCFS-EDS with revenue management becomes much higher after updating the booking limit (depicted in Table 6.8). By accepting too many discount-price users it rejects future booking of full-price users because resource capacity is limited, i.e. 40 compute nodes. This is called dilution

since it dilutes the revenue that could have received from saving additional compute nodes for full-price users. To anticipate future bookings that pay full price of compute nodes, it then limits the reservation from class 3 users from 30 to 0 compute nodes, and from class 2 users from 35 to 29 compute nodes as depicted in Table 6.8.

6.7 Revenue Management System strategy for MPI job

In the experiment in incorporating revenue management for MPI job, first, we have to make an experimental setup. Experimental setup for incorporating revenue management, for MPI jobs are:

1. We have 3 class of user, which we call premium user, business user and economy user.
2. Premium users also called as class 1 users pay p_1, i.e. full price for compute node.
3. Business users also called as class 2 users pay p_2, i.e. discount price for compute node.
4. Economy users also called as class 3 users pay p_3, i.e. discount price for compute node.
5. We set p_1= Rs. 100,- i.e. full price for 1 timeslot on one compute node, p_2= Rs. 60,- i.e. full price for 1 timeslot on one compute node, and p_3= Rs. 40,- i.e. full price for 1 timeslot on one compute node.

Table 6.9: Market Segmentation and Price Discrimination for MPI Job

Users	Book ahead	Price (Rs)
Class 1	$0 \leq ba \leq 12$	100
Class 2	$12 < ba \leq 24$	60
Class 3	$ba > 24$	40

6. Class 1 users reserve compute node between 0 to 12 timeslots in advance or book ahead is between 0 to 12 timeslots, class 1 users reserve compute node between 0 to 12 timeslots in advance, class 1 users reserve compute node

between 0 to 12 timeslots in advance. These market segmentation and price discrimination can be seen in Table 6.9.

7. Capacity of the compute node is 80 compute nodes.
8. Initial Booking Limit for class 1 users (b_1), class 2 users (b_2), and class 3 users (b_3) are depicted in Table 6.10. As depicted in Table 6.10, experiment #1 and #3 are designed to represent spoilage case, and experiment #2 and #4 are designed to represent dilution case.

We have done the experiment for our proposed advance planning and reservation scheduling strategy FCFS-EDS for parametric job. The workload or user requests of this experiment have these characteristics:

a. The rate of incoming reservation requests are assumed to follow poison distribution with mean as depicted in Table 6.10. If we use a very high rate of the incoming reservation then all computing nodes are reserved by class 1 users, which means there is no need to incorporate revenue management in it.
b. Execution time (t_e) for reservation requests are between 6 to 24 timeslots distributed uniformly.
c. Earlier starting time (t_{es}) for reservation requests are between 0 to 36 timeslots, distributed uniformly.
d. Percentage of user request that are for flexible advance reservation is assumed to at most 50% (selected randomly).
e. Relax time (t_r) for reservation requests are between 1 to 12 timeslots distributed uniformly and $t_{ls} = t_{es} + t_r$
f. Compute nodes needed are between 1 to 5 compute nodes distributed uniformly.
g. In the experiment it is assumed that a time slot is equal to 5 minutes (clock time).

In each experiment as depicted in Table 6.4, we have compared the revenue collected in sliding window of size 12 time slot (1 hour), between these three scenarios:

Table 6.10: Rate and Initial booking limit for the experiment

No	Rate	Initial b_1	Initial b_2	Initial b_3
1	2	80	20	10
2	2	80	75	70
3	3	80	20	10
4	3	80	75	70

1. Revenue of FCFS-EDS scheduling strategy without implementing revenue management. In Figure 6.22 to Figure 6.25 they are shown as blue line.
2. Revenue of FCFS-EDS with implementing revenue management without updating initial booking limit. In Figure 6.22 to Figure 6.25 they are shown as shown in red line.
3. Revenue of FCFE-EDS with implementing revenue management with updating booking limit. In Figure 6.22 to Figure 6.25 they are shown in green line. After updating the booking limit with EMSR-b formula, we have new initial booking limits as shown in Table 6.11 to Table 614.

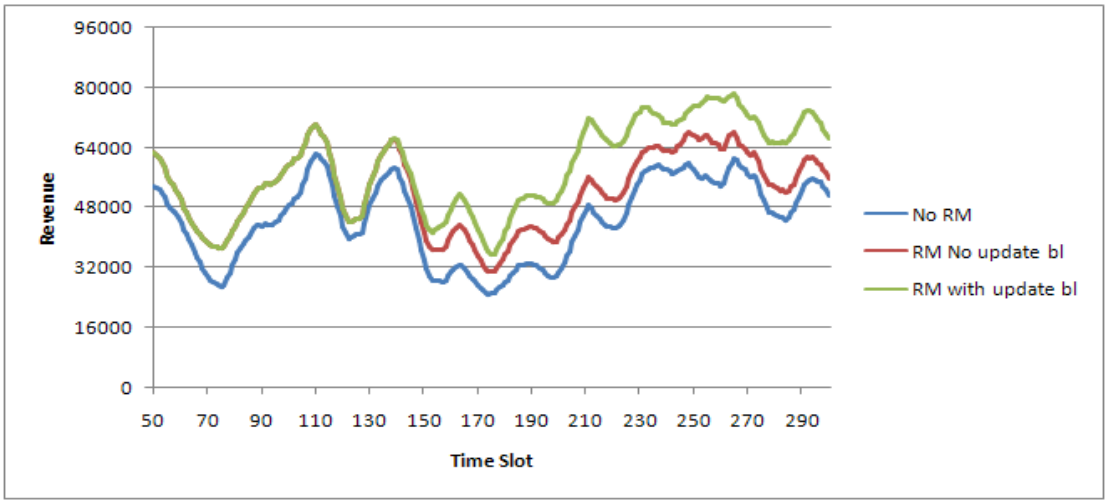

Figure 6.22: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking. (Experiment #1)

Table 6.11: Value of initial booking limit before and after update, for MPI Job (Experiment #1)

	Before update	After update
b_1	80	80
b_2	20	48
b_3	10	11

From Figure 6.22 (result of experiment #1), it can be seen that revenue of FCFS-EDS without revenue management has the lower revenue than that with revenue management, as it gains additional revenue by accepting discount-price users. This revenue becomes much higher after updating the booking limit (depicted in Table 6.11). By accepting less number of discount-price users, it hopes for future bookings that pay a full price of compute nodes. But then it may not see enough full-price users to reserve compute nodes. This is called spoilage as the idle compute nodes become spoiled. To reduce the spoilage, it then limits the reservation from class 3 users from 10 to 11 compute nodes, and from class 2 users from 20 to 48 compute nodes as depicted in Table. 6.11.

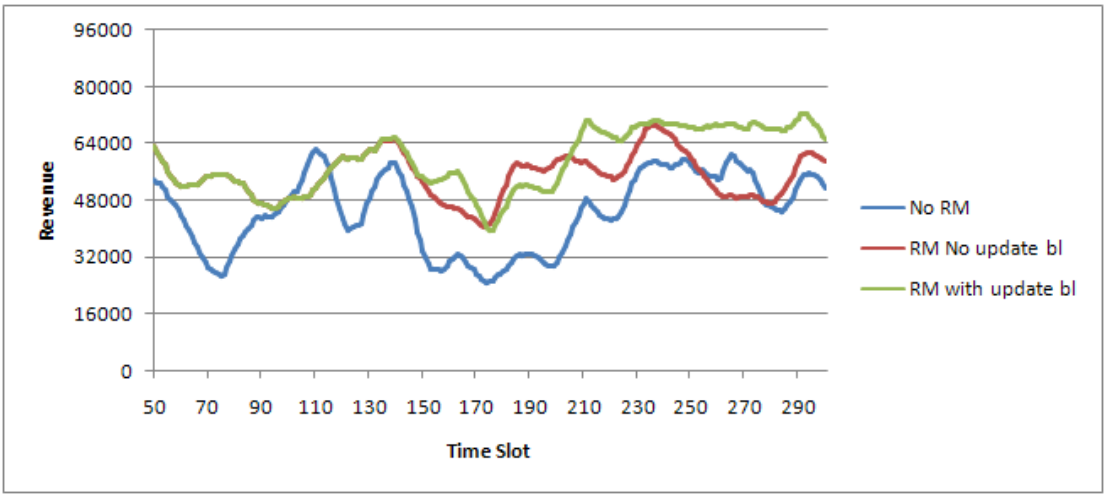

Figure 6.23: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #2)

Table 6.12: Value of initial booking limit before and after update for MPI Job (Experiment #2)

	Before update	After update
b_1	80	80
b_2	75	65
b_3	70	19

From Figure 6.23 (results of experiment #2) it can be seen that revenue of FCFS-EDS for MPI job without revenue management has lower revenue than that with revenue management, as it gains additional revenue by accepting discount-price users. This revenue becomes much higher after updating the booking limit (depicted in Table 6.12). By accepting too many discount-price users it rejects future booking of full-price users because resource capacity is limited, i.e. 80 compute nodes. This is called dilution since it dilutes the revenue that could have received from saving additional compute nodes for full-price users. To anticipate future bookings that pay full price of compute nodes, it then limits the reservation from class 3 users from 70 to 19 compute nodes, and from class 2 users from 75 to 65 compute nodes as depicted in Table 6.12.

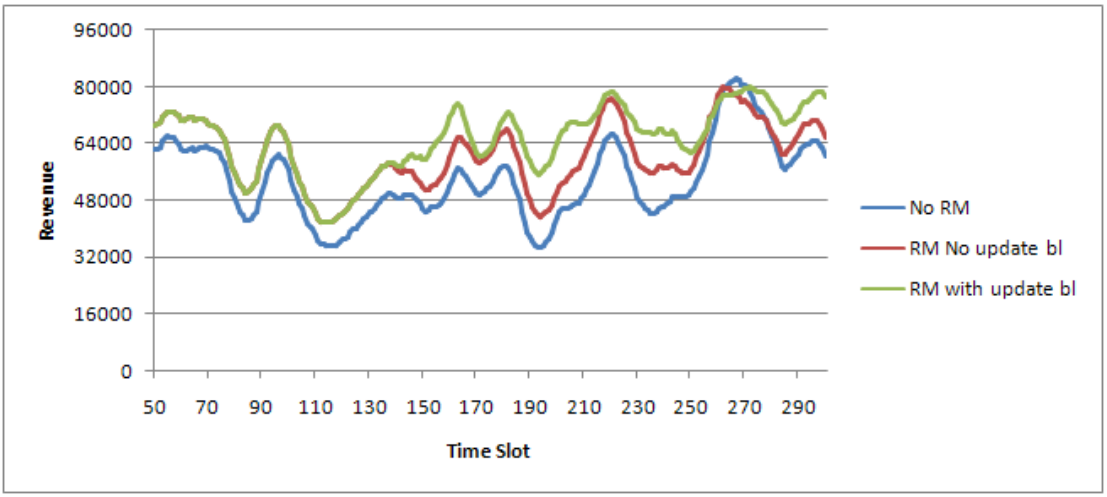

Figure 6.24: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #3)

Table 6.13: Value of initial booking limit before and after update for MPI Job (Experiment #3)

	Before update	After update
b_1	80	80
b_2	20	41
b_3	10	0

From Figure 6.24 (results of experiment #3), it can be seen that revenue of FCFS-EDS without revenue management has the lower than that with revenue management, as it gains additional revenue by accepting discount-price user. This revenue, even going higher, after updating the booking limit (depicted in Table 6.13). By accepting too less number of discount-price users, it hopes for future bookings that pay a full price of compute nodes. But then it may not see enough full-price booking to reserve compute nodes. This is called spoilage as the idle compute nodes become spoiled. To reduce the spoilage, it then limits the reservation from class 3 users from 10 to 0 compute nodes, and from class 2 users from 20 to 41 compute nodes as depicted in Table. 6.13.

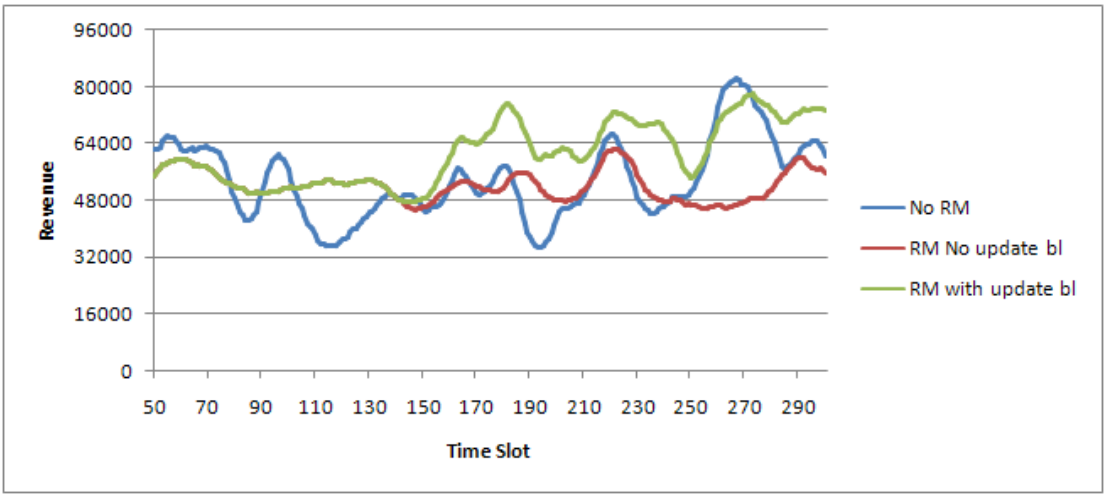

Figure 6.25: Comparison of revenue between FCFS-EDS scheduling strategy for MPI jobs without implementing revenue management, FCFS-EDS with implementing revenue management without updating initial booking limits, and FCFE-EDS with implementing revenue management with updating booking limits. (Experiment #4)

Table 6.14: Value of initial booking limit before and after update for MPI Job (Experiment #4)

	Before update	**After update**
$\boldsymbol{b_1}$	80	80
$\boldsymbol{b_2}$	75	62
$\boldsymbol{b_3}$	70	10

From Figure 6.25 (results of experiment #4), it can be seen that revenue of FCFS-EDS for MPI job with revenue management is lower than without revenue management, as it gains additional revenue by accepting discount-price users. The revenue of FCFS-EDS with revenue management becomes much higher after updating the booking limit (depicted in Table 6.12). By accepting too many discount-price users it rejects future booking of full-price user because resource capacity is limited, i.e. 80 compute nodes. This is called dilution. To anticipate future booking that pay full price of compute nodes, it then limits the reservation from class 3 users from 70 to 10 compute nodes, and from class 2 users from 75 to 62 compute nodes as depicted in Table 6.14.

7 Conclusions and Future Work

7.1 Conclusion

This thesis has attempted to broadly categorize the advance reservation strategies reported in the literature into: Rigid reservation [47] [48], Elastic Advance Reservation [43], Overlapping/Relax Advance Reservation [50] [54] [55], Flexible Advance Reservation (Static) [44] [58] [59], Flexible Advance Reservation (Dynamic, Physical View) [53] [63] [65]. Our proposed advance reservation scheduling strategy namely First Come First Serve Ejecting Based Dynamic Scheduling (FCFS-EDS) falls into Flexible Advance Reservation (Dynamic, Logical View).

This thesis has proposed a novel advance planning and reservation strategy namely First Come First Serve Ejecting Based Dynamic Scheduling (FCFS-EDS), which falls into Flexible Advance Reservation (Dynamic, Logical View), to increase resources utilization in a grid system. In the earlier scheduling like Flexible Advance Reservation (Dynamic, Physical View), necessary physical resources were reserved for the job and user was notified accordingly. If the physical resources were re-assigned a fresh notification had to be sent. In FCFS-EDS, it schedules the job at a logical level and if accepted the user is notified only once. Finally this thesis proposed a definition for flexible advance reservation: A new perception.

This thesis has compared the parameters of the proposed scheduling strategy FCFS-EDS with advance reservation scheduling strategies that have been used by other researchers. This thesis has proposed a new lemma (FCFS-EDS's lemma) to ensure "If there is a plan for scheduling a job on the consecutive time slot on virtual compute nodes (which are selected freely) (Logical View) then it guarantees that the job will be executed on a dedicated physical node for the required execution time (Physical View)". This thesis has also proposed a data structure, which falls into time slotted data structure to efficiently administer the proposed scheduling strategy.

To demonstrate the efficacy of the proposed FCFS-EDS strategy and the data structure, a novel simulator has been designed and implemented. This simulator has a potential to accommodate reservation for both Parametric as well as MPI job types. In this manner we have also generated the random input with a certain characteristic, such as the rate of incoming reservation following Poisson distribution, etc. This thesis has compared the performance of the proposed method (FCFE-EDS with advance planning) and an existing approach (flexible advance reservation strategy without advance planning) for parametric jobs. The comparison of resource utilization in both the strategies shows that FCFS-EDS yields better utilization than the traditional strategy (without advance planning). This thesis has also compared the performance of the proposed method FCFS-EDS with advance planning) and an existing approach (flexible advance reservation strategy without advance planning) for MPI jobs. The comparison of resource utilization in both the strategies shows that FCFS-EDS yields better utilization than the traditional strategy (without advance planning).

This thesis has incorporated revenue management in the proposed FCFS-EDS simulator to determine pricing of reservations in order to increase revenue/ throughput. This thesis has evaluated the effectiveness of revenue model and showed that by segmenting users, charging them with different price, and protecting resources for those who are willing to pay more, will result in increased revenue in both parametric jobs and MPI jobs. This thesis has also proposed architecture of the work as a complete system

7.2 Future Work

To enhance our work we suggest several future directions:

1. Incorporating pre-emptive jobs. Our proposed FCFS-EDS scheduling strategy can shift only reservations that are not yet executed. In future, FCFS-EDS strategy will be adapted so that already executing reservations can be pre-empted for a new one and later the pre-empted one can be resumed.

2. Incorporating other type of resources. In our work we only deal with reservation of compute nodes. For future work, we can incorporate heterogeneous compute nodes and other resources to be reserved such as storage resources and bandwidth resources. Some user may want to reserve a combination of any of those resources. In this case we have to update the reservation revenue management model to accommodate those additional resources.

3. Implementing the proposed advance planning and reservation strategy (FCFS-EDS) and revenue model in a real grid system such as Globus Toolkit or Sun Grid Engine.

4. Customizing the proposed advance planning and reservation strategy (FCFS-EDS) and revenue model in so that it can be adopted in Cloud System.

List of Publication

A) Journals

1. Rusydi Umar, Arun Agarwal, CR Rao, Data Structure for Advance Planning and Reservation in a Grid System, International Journal of Computer Applications (IJCA) ISSN: 09758887, Vol. NCRTC Issue: 1, pp. 33-37 (2012)

2. Rusydi Umar, Arun Agarwal, CR Rao, Advance Planning and Reservation for Parametric (MPI) jobs in a Cluster/Grid System, Journal of Networking Technology, ISSN: 0976-8998, Vol. 3, Issue: 3, pp. 129-138 (2012)

B) Conferences

1. Rusydi Umar, Arun Agarwal, CR Rao, Advance Planning and Reservation in a Grid System. In: Rachid Benlamri (eds.) NDT 2012. CCIS/LNCS, vol. 293, pp. 161-173. Springer, Heidelberg (2012)

References

[1] A. Oram, Peer-to-Peer: Harnessing the Power of Disruptive Technologies, California: O'Reilly Press, 2001.

[2] R. Buyya and A. Sulistiyo, "Service and Utility Oriented Distributed Computing Systems: Challenges and Opportunities for Modeling and Simulation Communities," in *Proceeding of the 41st Annual Simulation Symposium, 2008 (ANSS-41 '08), pp. 68-81. IEEE Computer Society*, Washington DC, 2008.

[3] J. Joshy and F. Craig, Grid Computing, New Jersey: Prentice Hall PTR, 2003.

[4] M. Daniel, A Networking Approach to Grid Computing, Hoboken, New Jersey: John Wiley & Sons, Inc, 2005.

[5] I. Foster and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, San Fransisco: Morgan Kaufmann, 1999.

[6] C. S. Yeo, R. Buyya, M. D. de Assuncao and J. Yu, Utility Computing and Global Grids, New York: John Wiley & Sons, 2007.

[7] W. Cirne, D. Paranhos, L. Costa and E. Santos-Neto, "Running Bag-of-Tasks applications on computational grids: the MyGrid approach," in *The 32nd International Conference on Parallel Processing, 2003 (ICPP 2003), pp. 407-416. IEEE Computer Society*, Kaohsiung, Taiwan , 2003.

[8] D. Abramson, J. Giddy and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?," in *The 14th International Parallel and Distributed Processing Symposium (IPDPS'00), pp. 520-528, IEEE Press*, Cancun, 2000.

[9] L. B. Costa, L. Feitosa, E. Araujo, C. Mendes, R. Coelho, W. Cirne and D.

Fireman, "MyGrid: A complete solution for running bag-of-tasks applications," in *22nd Brazilian Symposium on Computer Networks (SBRC'04), IEEE Press* , Brazil, 2004.

[10] R. Buyya, D. Abramson and J. Giddy, "Architecture for a Resource Management and Scheduling System in a Global Computational Grid," in *4th International Conference & Exhibition on High Performance Computing in Asia-Pacific Region (HPC Asia'00), pp. 283-289, IEEE Press*, Beijing, 2000.

[11] D. P. Anderson, J. Cobb , E. Korpela, M. Lebofsky and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56-61, 2002.

[12] S. Graupner, J. Pruyne and S. Singhal , "Making the Utility Data Center a Power Station for the Enterprise Grid," Hewlett-Packard Laboratories, Internet and Computing Platforms Research Center, Palo Alto, 2003.

[13] R. Buyya and S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report," in *1st International Workshop on Grid Economics and Business Models (GECON'04), pp. 19-66, IEEE Press*, Seoul, 2004.

[14] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Network and Computer Applications,* vol. 23, pp. 187-200, 2001.

[15] W. Hoschek, F. J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger, "Data management in an international data grid project," in *1st International Workshop on Grid Computing (Grid'00), pp. 77-79, Springer-Verlag*, London, 2000.

[16] J. C. Jacob, D. S. Katz, T. Prince , G. B. Berriman, J. C. Good, A. Laity, C. E. Deelman, G. Singh and M. H. Su, "The Montage Architecture for Grid-enabled Science Processing of Large, Distributed Datasets," in *The fourth annual Earth Science Technology Conference*, Palo Alto, 2004.

[17] M. J. Mineter, C. H. Jarvis and S. Dowers, "From stand-alone programs towards

grid-aware services and components: a case study in agricultural modeling with interpolated climate data," *Environmental Modelling and Software,* vol. 18, no. 4, pp. 379-391, 2003.

[18] Avaki EII - Enterprise Data Integration Sofware, [Online]. Available: http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.avaki_7.0/title.htm.

[19] Biogrid Project, [Online]. Available: http://biocodenv.com/biogridarticles/53-biogrid.

[20] International Virtual Observatory Alliance, [Online]. Available: http://www.ivoa.net.

[21] LCG Computing Fabric Area, [Online]. Available: http://lcg-computing-fabric.web.cern.ch.

[22] EU Data Mining Grid, [Online]. Available: http://www.datamininggrid.org.

[23] M. Cannataro and D. Talia, "The Knowledge Grid," *Communications of the ACM,* vol. 46, no. 1, pp. 89-93, 2003.

[24] K. Seymour, A. YarKhan, S. Agrawal and J. Dongarra, "NetSolve: Grid Enabling Scientific Computing Environments," *Grid Computing and New Frontiers of High Performance Processing,* vol. 14, pp. 33-51, 2005.

[25] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens and T. Udeshi, "Access Grid: Immersive Group-to-Group Collaborative Visualization," in *The 4th International Immersive Projection Technology Workshop,* Ames, USA, 2000.

[26] M. Chetty and R. Buyya, "Weaving Computational Grids: How Analogous Are They With Electrical Grids?," in *IEEE Computing in Science and Engineering, pp. 61-71. IEEE Computer Society*, Los Alamitos, USA, 2002.

[27] B. Jacob, M. Brown, K. Fukui and N. Trivedi, Introduction to Grid Computing, USA: IBM Corporation , 2005.

[28] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Girard, R. Strachowski and S. Yu, Enabling Applications for Grid Computing with Globus, USA: IBM Corporation , 2003.

[29] I. Foster and C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, Los Altos, California: Morgan Kaufmann, 2004.

[30] A. Z. Isah and H. Safwana, "Resource Management in Grid Computing: A Review," *Greener Journal of Science Engineering and Technological Research*, vol. 2, no. 1, pp. 24-31, 2012.

[31] R. Buyya, D. Abramson and J. Giddy, "Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid," in *The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), pp. 283-289. IEEE Computer Society Press*, Beijing, 2000.

[32] R. Buyya, D. Abramson and J. Giddy, "An Economy Driven Resource Management Architecture for Global Computational Power Grids," in *The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), CSREA Press*, Las Vegas, USA, 2000.

[33] I. Foster, A. Roy and V. Sander, "A quality of service architecture that combines resource reservation and application adaptation," in *8th International Workshop on Quality of Service (IWQOS 2000), pp. 181-188, IEEE Press*, Pittsburgh, 2000.

[34] Y. Juan, B. Yun and Q. Yuhui, "A decentralized resource allocation policy in minigrid," *Future Generation Computer Systems*, vol. 23, no. 3, pp. 359-366, 2007.

[35] M. S. Jennifer, "Ten Actions When Grid Scheduling: The User as a Grid Scheduler," in *Grid Resource Management: State of the Art and Future Trends*, Norwell, MA, USA, Kluwer Academic Publishers, 2004, pp. 15-24.

[36] S. Uwe and Y. Ramin, "Attributes for Communication Between Grid Scheduling Instances," in *Grid Resource Management: State of the Art and Future Trends*, Norwell, MA, USA, Kluwer Academic Publishers, 2004, pp. 41-52.

[37] U. Rusydi, A. Arun and C. R. Rao, "Data Structure for Advance Planning and Reservation in a Grid System," *International Journal of Computer Applications (IJCA)*, vol. NCRTC, no. 1, pp. 33-37, 2012.

[38] A. W. Mu'alem and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE*

Transactions on Parallel and Distributed Systems, vol. 12, pp. 529-543, 2001.

[39] A. Sulistio and R. Buyya , "A Grid simulation infrastructure supporting advance reservation," in *16th International Conference on Parallel and Distributed Computing and Systems, pp. 1–7. ACTA Press*, Calgary, 2004.

[40] J. MacLaren, "Advance Reservations: State of the Art," Working Draft, Global Grid Forum , 2003.

[41] W. Smith, I. Foster and V. Taylor, "Scheduling with Advanced Reservations," in *14th IEEE International Symposium on Parallel and Distributed Processing, pp. 127–132. IEEE Press,* Cancun, 2000.

[42] R. Buyya, D. Abramson and S. Venugopal, "The Grid Economy," *Proceedings of The IEEE,* vol. 93, no. 3, pp. 698-714, 2005.

[43] A. Sulistio , K. H. Kim and R. Buyya , "On Incorporating an On-line Strip Packing Algorithm into Elastic Grid Reservation-based Systems," in *13th International Conference on Parallel and Distributed Systems (ICPADS'07), pp. 1-8, IEEE Press*, Hsinchu, 2007.

[44] C. Moaddeli, H. R. Dastghaibyfard and G. R. Moosavi, "Flexible Advance Reservation Impact on Backfilling Scheduling Strategies," in *7th International Conference on Grid and Cooperative Computing, pp. 151-159, IEEE Press*, Shenzhen, 2008.

[45] M. Siddiqui, A. Villazon and T. Fahringer, "Grid capacity planning with negotiation-based advance reservation for optimized QoS," in *the 2006 ACM/IEEE conference on Supercomputing (SC'06) pp. 21*, Florida, 2006.

[46] R. L. Phillips, Pricing and Revenue Optimization, Stanford: Stanford University Press, 2005.

[47] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation," in *7th IEEE International Workshop on Quality of Service, pp. 27-36, IEEE Press*, London, 1999.

[48] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith and S.

Tuecke, "A resource management architecture for metacomputing systems," in *4th Workshop on Job Scheduling Strategies for Parallel Processing. LNCS vol. 1459, pp 62-82. Springer*, London, 1998.

[49] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distri-buted Management and Scheduling for Grid Computing," *Concurrency and Computation: Practice and Experience (CCPE),* vol. 14, no. 13, pp. 1175-1220, 2002.

[50] P. Xiao, H. Zhigang, L. Xi and Y. Liu, "A Novel Statistic-based Relaxed Grid Resource Reservation Strategy," in *9th International Conference for Young Computer Scientists, pp. 703-707. IEEE Press*, Hunan, 2008.

[51] C. B. Lee and A. Snavely, "On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions," *International Journal of High Performance Computing Applications,* vol. 20, pp. 496-506, 2006.

[52] C. Castillo, G. Rouskas and K. Harfoush, "On the design of online scheduling algorithms for advance reservations and QoS in grids," in *Parallel and Distributed Processing Symposium, IPDPS, IEEE International, pp. 1-10*, Long Beach, USA , 2007.

[53] H. Chunming, H. Jinpeng and W. Tianyu , "Flexible Resource Reservation Using Slack Time for Service Grid," in *12th International Conference on Parallel and Distributed Systems, pp. 327-334, IEEE Press*, Washington , 2006.

[54] X. Peng and H. U. Zhigang, "Relaxed resource advance reservation policy in grid computing," *The Journal of China Universities of Posts and Telecommunications,* vol. 16, pp. 108-113, 2009.

[55] R. B. S. Sabitha, R. Venkatesan and R. Ramalakshmi, "Resource Reservation In Grid Computing Environments: Design Issues," in *3rd International Conference on Electronics Computer Technology, pp. 66-70, IEEE Press*, Kanyakumari, 2011.

[56] D. A. Lifka, "The ANL/IBM SP scheduling system," in *the 1st International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP). LNCS, vol. 949, pp. 295-303, Springer*, Santa Barbara , 1995.

[57] D. Feitelson and A. Weil, "Utilization and Predictability in Scheduling the IBM SP2 with Back," in *The 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP-98), pages 542–547*, Los Alamitos, 1998.

[58] N. R. Kaushik, S. M. Figueira and S. A. Chiappari, "Flexible Time-Windows for Advance Reservation Scheduling," in *14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 218-225, IEEE Press*, California , 2006.

[59] C. Castillo, G. Rouskas and K. Harfoush, "Online algorithms for advance resource reservations," *Journal of Parallel and Distributed Computing,* vol. 71, pp. 963-972, 2011.

[60] M. de Berg, M. van Krefeld, M. Overmars and O. Cheong, Computational Geometry: Algorithms and Applications, Netherland: Springer-Verlag, 2000.

[61] T. D. Braun, H. J. Siegal, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, Y. Bin, D. Hensgen and R. F. Freund, "A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems," in *8th IEEE Heterogeneous Computing Workshop (HCW'99), pp. 15-29, IEEE Press*, San Juan , 1999.

[62] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environment," in *the 9th Heterogeneous Computing Workshop (HCW'2000), pp. 349-363, IEEE Press*, Cancun, 2000.

[63] M. A. S. Netto, K. Bubendorfer and R. Buyya, "SLA-based advance reservations with flexible and adaptive time QoS parameters," in *5th International Conference on Service-Oriented Computing, LNCS vol. 4749, pp. 119-131. Springer* , Chicago, 2007.

[64] Y. L. Wu, W. Huang, S. C. Lau, C. K. Wong and G. H. Young, "An effective quasi-human based heuristic for solving the rectangle packing problem," *European Journal of Operational Research,* vol. 141, no. 2, pp. 341-358, 2002.

[65] B. Behnam, M. R. Amir, Z. F. Kamran and D. Azedah, "Gravitational Emulation

Local Search Algorithm for Advanced Reservation and Scheduling in Grid Computing Systems," in *4th International Conference on Computer Science and Convergence Information Technology, pp. 1240-1245, IEEE Press*, Seoul, 2009.

[66] L. W. Barry, "Solving combinatorial optimization problems using a new algorithm based on gravitational attraction," Ph.D. Thesis, Melbourne, Florida Institute of Technology, Florida, 2004.

[67] L. -O. Burchard, "Analysis of data structures for admission control of advance reservation requests," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 413 - 424, March 2005.

[68] L. -O. Burchard and H. -U. Heiss, "Performance Evaluation of Data Structures for Admission Control in Bandwidth Brokers," in *International Symposium of Performance Evaluation of Computer and Telecommunication Systems (SPECTS '02), pp. 652-659*, San Diego, 2002.

[69] R. Guerin and A. Orda, "Networks with Advance Reservations: The Routing Perspective. In Proceeding of the Conference on Computer Communications," in *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Reaching the Promised Land of Communications (INFOCOM 2000), pp. 118-127, IEEE Press*, Tel Aviv , 2000.

[70] O. Schelen, A. Nilsson, J. Norrgard and S. Pink , "Performance of QoS Agents for Provisioning Network Resources," in *Seventh International Workshop on Quality of Service (IWQoS '99), pp. 17-26, IEEE Press*, London, 1999.

[71] A. Brodnik and A. Nilsson, "A static data structure for discrete advance bandwidth reservations on the internet," in *Swedish National Computer Networking Workshop (SNCNW)*, Stockholm , 2003.

[72] R. Brown, "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem," *Communications of the ACM*, vol. 31, no. 10, pp. 1220-1227, 1988.

[73] A. Sulistio, U. Cibej, S. Prasad and R. Buyya, "GarQ: An Efficient Scheduling Data Structure for Advance Reservations of Grid Resources," *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, vol. 24, no. 1, pp. 1-19,

2008.

[74] T. Wang and J. Chen, "Bandwidth tree – a data structure for routing in networks with advanced reservations," in *21st Interrnational Performance, Computing, and Communications Conference (IPCCC), pp. 37–44, IEEE Press*, Phoenix , 2002.

[75] L. Yuan, C. -K. Tham and A. L. Ananda, "A probing approach for effective distributed resource reservation," in *the 2nd International Workshop on Quality of Service in Multiservice IP Networks, pp. 672–688, Springer-Verlag*, Milan , 2003.

[76] K. Melhorn, Data structures and algorithms III: Multi-dimensional searching and computational geometry, New York : Springer-Verlag, 1984.

[77] Q. Xiong, C. Wu, J. Xing, L. Wu and H. Zhang, "A linked-list data structure for advance reservation admission control," in *the 3rd International Conference on Networking and Mobile Computing (ICCNMC). LNCS, vol. 3619, pp. 901-910*, Zhangjiajie, China, 2005.

[78] W. Tao and C. Jianer, "Bandwidth Tree - A Data Structure for Routing in Networks with Advanced Reservations," in *21st IEEE International Conference on Performance, Computing, and Communications (IPCCC 2002), pp. 37-44, IEEE Press*, Phoenix, 2002.

[79] S. J. Chapin, K. Dimitios, J. F. Karpovich and A. S. Grimshaw, "The legion resource management system," in *5th Workshop of Job Scheduling Strategies for Parallel Processing. LNCS, vol. 1659, pp. 162-178*, San Juan, Puerto Rico, 1999.

[80] M. Litzkow, M. Livny and M. Mutka, "Condor - A hunter of idle workstations," in *8th International Conference on Distributed Computing Systems (ICDCS 1988), pp. 104-111, IEEE Press*, San Jose, 1988.

[81] F. Berman and R. Wolski, "The AppLeS project: A status report," in *The 8th Heterogeneous Computing and Multidisciplinary Applications NEC Research Symposium, pp. 1-21*, Berlin, 2000.

[82] H. Casanova, G. Obertelli, F. Berman and R. Wolski, "The AppLeS parameter sweep template: User-level middleware for the grid," in *The IEEE Supercomputing Conference (SC 2000), pp. 60, IEEE Press*, Dallas, 2000.

[83] H. Casanova and J. Dongarra, "NetSolve: A network server for solving computational science problems," *The International Journal Supercomputing Application and High Performance Computing*, vol. 11, no. 3, pp. 212-223, 1997.

[84] N. Kapadia and J. Fortes, "PUNCH: An architecture for web-enabled wide-area network-computing," *Cluster Computing*, vol. 2, no. 2, pp. 153-164, 1999.

[85] G. Fedak, C. Germain, V. Néri and F. Cappello, "XtremWeb : A generic global computing system," in *The 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), pp. 582-587, IEEE Press*, Brisbane, 2001.

[86] R. Cocchi, S. Shanker, D. Estrin and L. Zhang, "Pricing in computer networks: Motivation, formulation, and example," *IEEE/ACM Transaction on Networking*, vol. 1, no. 6, pp. 614-627, 1993.

[87] T. Roeblitz, F. Schintke and A. Reinefeld, "Resource reservations with fuzzy requests," *Concurrency and Computation: Practice & Experience (CCPE)*, vol. 18, no. 13, pp. 1681-1703, 2006.

[88] G. Pfister, In Search of Clusters, New Jersey: Prentice Hall PTR, NJ, 2nd Edition, 1998.

[89] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, pp. 114-117, 1965.

[90] K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming, New York: WCB/McGraw-Hill, 1998.

[91] P. T. Matthew and S. K. W. Johnny, "A Task Migration Algorithm for Heterogeneous Distributed Computing Systems," *System and Software*, vol. 41, pp. 175-188, 1998.

[92] U. Rusydi, A. Arun and C. R. Rao, "Advance Planning and Reservation in a Grid System," in *NDT 2012. CCIS/LNCS, vol. 293, pp. 161-173. Springer, Heidelberg* , Dubai, 2012.

[93] U. Rusydi, A. Arun and C. R. Rao, "Advance Planning and Reservation for Parametric (MPI) jobs in a Cluster/Grid System," *Journal of Networking Technology*, vol. 3, no. 3, pp. 129-138, 2012.

[94] R. Lawrence, Weatherford and E. B. Samuel, "A taxonomy and research overview of perishable-asset revenue management: yield management, overbooking, and pricing," *Operations Research,* vol. 40, no. 5, pp. 831-844, 1992.

[95] A. Anandasivam and D. Neumann, "Managing Revenue in Grids," in *42nd Hawaii International Conference (HICSS) on System Sciences, pp.1-10, IEEE Press*, Hawaii, 2009.

[96] S. Nair and R. Bapna , "An Application of Yield Management for Internet Service Providers," *Naval Research Logistics,* vol. 48, pp. 348-362, 2001.

[97] B. Gabriel and C. Rene, "An overview of pricing models for revenue management," *Manufacturing & Service Operations Management,* vol. 5, no. 3, pp. 203-229, 2003.

[98] S. Kimes, "Yield management: A tool for capacity-constrained service firms," *Journal of Operations Management,* vol. 8, no. 4, pp. 348-363, 1989.

[99] M. Armstrong, "Price discrimination," MPRA Paper, Department of Economics, University College London, London, 2006.

[100] K. Littlewood, "Forecasting and control of passenger bookings," *Journal of Revenue and Pricing Management,* vol. 4, pp. 111-123, 2005.

[101] S. L. Brumelle, J. I. McGill, T. H. Oum, K. Sawaki and M. W. Tretheway, "Allocation of airline seats between stochastically dependent demands," *Transportation Science,* vol. 24, no. 3, pp. 183-192, 1990.

[102] K. T. Talluri and G. J. V. Ryzin, The Theory and Practice of Revenue Management, Boston: Kluwer Academic Publisher, 2005.

[103] P. P. Belobaba, "Air Travel Demand and Airline Seat Inventory Management," PhD thesis, Flight Transportation Laboratory, MIT, Cambridge, 1987.

[104] P. P. Belobaba, "Optimal vs. heuristic methods for nested seat allocation," ORSA/TIMS Joint National Meeting , 1992.

[105] T. K. Talluri and G. J. van Ryzin, "Revenue management under a general discrete choice model of consumer behavior," *Management Science,* vol. 50, no. 1, pp. 15-33, 2004.

[106] R. D. Wollmer, "An airline seat management model for a single leg route when lower fare classes book first," *Operations Research*, vol. 40, pp. 26-37, 1992.

[107] A. Sulistio, K. H. Kim and R. Buyya, "Using Revenue Management to Determine Pricing of Reservations," in *the 3rd International Conferenceon e-Science and Grid Computing (e-Science'07), pp. 396-405, IEEE Press*, Bangalore, 2007.

[108] A. Sulistio, K. H. Kim and R. Buyya, "Managing Cancellations and No-shows of Reservations with Overbooking to Increase Resource Revenue," in *the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08), pp. 267-276* , Lyon, 2008.