Resource Allocation Strategies for Multiple Job Classes

by

Ye Hu

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 2009

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Resource management for a data center with multiple job classes is investigated in this thesis. We focus on strategies for allocating resources to an application mix such that the service level agreements (SLAs) of individual applications are met. A performance model with two interactive job classes is used to determine the smallest number of processor nodes required to meet the SLAs of both classes. For each class, the SLA is specified by the relationship: Prob $(response\ time \leq x) \geq y$. Two allocation strategies are considered: shared allocation (SA) and dedicated allocation (DA). For the case of FCFS scheduling, analytic results for response time distribution are used to develop a heuristic algorithm that determines the allocation strategy (SA or DA) that requires fewer processor nodes. The effectiveness of this algorithm is evaluated over a range of operating conditions. The performance of SA with non-FCFS scheduling is also investigated. Among the scheduling disciplines considered, a new discipline called probability dependent priority (PDP) is found to have the best performance in terms of requiring the smallest number of nodes. Furthermore, we extend our heuristic algorithm for FCFS to three job classes. The effectiveness of this extended algorithm is evaluated. As to priority scheduling, the performance advantage of PDP is also confirmed for the case of three job classes.

Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor, Professor Johnny Wong, whose expert guidance and experience directed me through my Master studies. This work would not be made possible without his encouragement and precious advice. His wide knowledge and logical way of thinking have been of great value for me. Thanks given to Professor Ashraf Aboulnaga and Raouf Boutaba for taking time to be the readers of this thesis.

I want to thank my friends and officemates for brining pleasant and enjoyable moments to me during my study. Finally, I express my sincere thanks to my parents and family for their understanding, love and support.

Contents

List of Tables **vii**

List of Figures **viii**

1. Introduction **1**

2. Survey of related work **7**

2.1 Queueing theory 7

2.2 Control theory 8

2.3 Machine learning approach 8

2.4 Market-based approach 9

2.5 Hybrid approach 10

3. Performance model **11**

4. Allocation strategies for two interactive classes **14**

4.1 Analytic results for FCFS 14

4.2 Allocation strategies 15

4.3 DA and SA comparison 17

4.4 Heuristic algorithm for two interactive classes 18

4.4.1 SLA difference 18

4.4.2 Heuristic algorithm 24

4.5 Performance evaluation 27

4.5.1 Methodology 27

4.5.2 Performance results 29

5. Priority disciplines **31**

5.1 Head of the line priority 31

5.2 Probability dependent priority ……..… 32
5.3 Performance evaluation ……. 33
6. Allocation strategies for three interactive classes 36
6.1 Allocation strategies …….. 37
6.2 Heuristic algorithm for three interactive classes ……..……. 38
6.3 Performance evaluation ……..…….. 42
6.4 Probability dependent priority scheduling …… 44
7. Conclusion and future work 47
7.1 Conclusion ……..…….. 47
7.2 Future work ……..…… 48
References 50

List of Tables

4.1 Parameter values of arrival rates and SLAs 17

4.2 Two example cases 18

4.3 SLA pairs where $D = 22.6$ 21

4.4 SLA pairs where $D = 83.5$ 21

4.5 Angle table 26

4.6 Probability distributions 28

4.7 Probability of correct strategy: Algorithm 2 30

5.1 Performance comparison: 2 job classes 35

5.2 Performance difference: 2 job classes 35

6.1 Performance of the four solutions 41

6.2 Probability of correct strategy: Algorithm 3 44

6.3 Performance comparison: 3 job classes 46

6.4 Performance difference: 3 job classes 46

List of Figures

1.1 Two-level architecture 2

3.1 Shared allocation 12

3.2 Dedicated allocation 12

4.1 Number of processor nodes required 19

4.2 Scenario 1 22

4.3 Scenario 2 22

4.4 Scenario 3 23

4.5 Scenario 4 23

4.6 Percentage of DA vs. SLA difference 24

4.7 Heuristic method 25

4.8 Angle vs. SLA difference 25

6.1 Resource allocation graph structures 39

6.2 Solutions to allocation structure 3 40

Chapter 1

Introduction

To meet the increasing demand for computing resources, the size and complexity of today's data center are growing rapidly. At the same time, technologies like server clusters, grids, and cloud computing are becoming more popular. An immediate question is how the resources in a data center may be managed in a cost-effective manner. Static resource allocation based on peak demand is not cost-effective because of poor resource utilization during off-peak time periods. In contrast, autonomic resource management could lead to efficient resource usage and fast response in the presence changing workloads.

Autonomic resource management has received considerable attention in recent years. Topics investigated include:

- Self-optimization – optimizes the resource allocation and seeks performance improvement opportunities.
- Self-healing – detects, diagnoses and recovers from failures.
- Self-configuration – re-configures the system according to high-level objectives.

- Self-protection – prevents the system from attacks and crashes.
- Power saving – reduces energy usage and cooling cost.

This thesis is concerned with resource allocation strategies that are relevant to autonomic resource management in achieving self-optimization.

The two-level resource management architecture presented in [1] provides a framework for our investigation, as shown in Figure 1.1. This architecture has two levels. At the lower level, there are multiple application environments (AEs). Each AE consists of a set of computing resources that are shared by one or more applications. At the higher level, a global arbiter performs resources allocation across AEs.

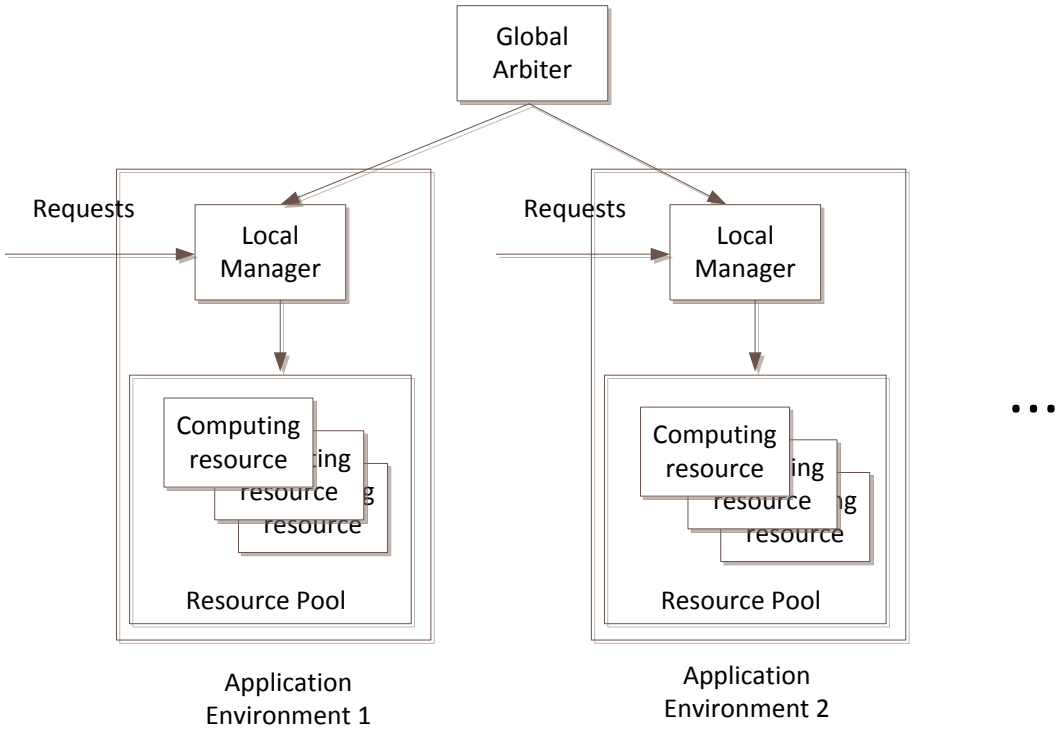

Figure 1.1: Two-level architecture

Jobs executed at a data centre can broadly be categorized as interactive or batch. Interactive jobs generally have small processing requirements and require good response time performance. Batch jobs, on the other hand, are usually long-running and the performance metrics of interest are throughput and the percentage of jobs that are completed on time. In this thesis, we only consider the processing of interactive jobs, e.g., web-based systems and multi-user online games. Multiple interactive job classes are considered where each class may have its own workload and service level agreement (SLA). In our investigation, the SLAs are based on the response time distribution, namely, Prob $(response\ time \leq x) \geq y$ where x is a threshold value and y is the target probability. We aim to obtain results that can be used to guide resource allocation decisions. These results are derived from solutions to a performance model.

In our investigation, computing resources at each AE are modeled by processor nodes. When the global arbiter makes resource allocation decisions, information on the number of processor nodes that should be allocated to each AE would be very helpful. This corresponds to the smallest number of processor nodes required to meet the SLAs of those applications that are assigned to the AE. In this thesis our focus is on resource management within an AE.

Jobs processed by an AE are classified according to their workloads and SLAs. One or more applications may be included in the same class. The smallest number of processor nodes mentioned above is affected by the resource allocation strategy and job scheduling discipline within the AE. The resource allocation strategies under

consideration are shared allocation (SA) and dedicated allocation (DA). In SA, the processor nodes are shared by all job classes with no preferential treatment on the basis of class membership. DA, on the other hand, allocates to each job class a fixed number of processor nodes; these processors are not available to other classes.

The performance seen by the different job classes is affected by the scheduling discipline used. The disciplines considered include first-come first-served (FCFS) where no preferential treatment is given to jobs belonging different classes, and two priority disciplines where job classes with more demanding SLAs are given higher priority.

In [2], a multi-server queueing model was used to show that SA is superior to DA with respect to mean response time over all jobs. However, the issue of meeting SLAs was not included in the investigation. When SLAs are considered, SA may not be the better strategy under all operating conditions.

In general, an AE may have a potentially large number of job classes. Results on the performance difference between SA and DA for an arbitrary number of classes may be difficult to obtain. This is because of the potentially large number of possible allocation strategies that need to be evaluated. Additional complexity is introduced when the impact of scheduling discipline is included in the investigation.

To keep the complexity at a modest level, we start with the special case of two interactive job classes. Our investigation includes (i) a comparative evaluation of SA and DA under FCFS scheduling; (ii) a heuristic algorithm that determines a resource allocation strategy (SA or DA) that results in the smallest number of processor nodes

required to meet the SLAs of both classes; and (iii) a comparative evaluation of FCFS, head-of-the-line priority (HOL) and a new scheduling discipline called probability dependent priority (PDP). Our results provide valuable insights into the performance of alternative resource allocation strategy and job scheduling disciplines. These results can also be used to develop guidelines for resource management when there are more than two classes.

This thesis makes the following contributions:

1. An important finding that SA is not always the better resource allocation strategy compared to DA with respect to response time distribution when SLA is taken into consideration.
2. For FCFS scheduling, a heuristic algorithm that determines a resource allocation strategy that results in the smallest number of processor nodes required for the case of two job classes, as well as an extension of this algorithm to three job classes.
3. The development of a new scheduling algorithm (called PDP) that is superior in performance when compared to FCFS and HOL.

The remainder of this thesis is organized as follows. Chapter 2 gives a survey of existing work on dynamic resource provisioning. Our performance model is described in Chapter 3. Chapter 4 presents results on the merits of SA and DA under FCFS. A heuristic algorithm to select the preferred resource allocation strategy under FCFS is also developed and evaluated. The impact of priority scheduling on performance is

investigated in Chapter 5. Chapter 6 extends the results to the case of three job classes. Finally, Chapter 7 contains a summary of our findings and a discussion of topics for future research.

Chapter 2

Survey of related work

Related work in dynamic resource management can be organized according to the approach used in the investigation, including queueing theory, control theory, machine learning, market-based approach and hybrid approach.

2.1 Queueing theory

Queuing theory [3-8] is a well-established and widely used methodology in performance evaluation of resource management strategies due to its ability in performance prediction. Performance results are often used to guide resource allocation decisions. In [8], the authors present utility models based on a system of multiple parallel M/M/1 queues to study a Trade3 application, which is a realistic representation of an electronic trading platform. The mean response time and throughput from the M/M/1 models are used to maximize the total utility. In [3], a multiclass queueing network model is used to compute the mean response time. A layered queueing network is used in [4, 5] to study the effect of workload and the system parameters on performance. A regression-based

approximation of the CPU demand of client transactions is introduced in [6]; the approximation is obtained using a network of queues model with each queue representing an application tier.

Despite its spread use, queueing theory has some limitations. These include the need to make potentially unrealistic assumptions in order to obtain analytic results and solutions for complex models may be difficult to obtain.

2.2 Control theory

Control theory [9-12] has been used in the design of dynamic resource management schemes because of properties such as self-correcting and self-stabilizing. In [9], a system is developed that can meet application-level quality of service while achieving high resource utilization. An analytic foundation of control theory for a self-managing system is described in [10]. In [11], the authors argue that control theory should be used to build and to configure self-managing systems. The 1000 Island solution architecture is presented in [12]; this architecture has multiple resource controllers that are based on control theory and optimization methods.

2.3 Machine learning approach

Machine learning has also been used in autonomic resource management [13-16]. In [13], an off-line proactive learning approach called K-nearest-neighbours is proposed to

dynamically allocate database replicas. A lightweight on-line learning of correlations between system state and response time is described in [14]. In [15], an active learning approach is used to build models to predict the completion time of batch jobs. A combination of off-line reinforcement learning and queueing theory is used to improve the performance prediction [16].

The effectiveness of learning methods largely depends on the training set which may not be easy to obtain. Also, a long training period may not be desirable, especially in the case for on-line learning.

2.4 Market-based approach

Market-based approaches [17-19] allow applications to specify their utility in terms of quality of service guarantees. There are market agents who know how to transform quality of service requirements into actual resources and how to trade extra resources between applications. In [17], the price-directed idea is used to address admission control and resource allocation problems in integrated-services networks. In another study, a free market approach is presented where each application can trade its computing resources with others according to some market policies [18]. In this approach, the marketplace determines a price for each unit of resource and reallocates resource by moving resources from sellers to buyers. The authors claim that their approach is able to effectively provision resources at both stable and unstable states.

It has been found that the market-based approach works well when mappings between resource and quality of service guarantees can be established. However, determining these mappings is often difficult in practice.

2.5 Hybrid approach

A hybrid approach [20, 21] usually benefits from the good properties of two or more approaches. For example, queueing theory is useful in performance prediction and feedback control can provide self-correcting and self-stabilizing behaviours. These two approaches are combined in [20, 21] to achieve quality of service support in highly unpredictable environments. In that method, a feedback control loop compares the measured delay with the desired average and then adjusts the resource allocation in an incremental manner to ensure that the desired delay is maintained. Another example is the method that combines queueing theory and statistical learning mentioned in Section 2.3 [16].

Chapter 3

Performance model

In our model, the computing resources at each AE are modeled by processor nodes. These nodes process jobs according to a given scheduling discipline. Multiple interactive job classes are considered where each job class has its own workload and SLA. For the case of two job classes, the number of AEs is either 1 or 2 and the corresponding resource allocation strategies are SA or DA. Our models for SA and DA are shown in Figures 3.1 and 3.2, respectively. For SA, job arrivals from the two classes are combined into a single stream and served by a pool of m processor nodes. For DA, each job class has its own dedicated pool of processor nodes, and we use m_1 and m_2 to denote to number of processor nodes allocated to class 1 and class 2, respectively.

We assume that for class i $(i = 1,2)$, the arrival process is Poisson with rate λ_i and the service time distribution of both classes is exponential with mean $1/\mu$. As mentioned earlier in the introduction, the SLA is based on the relationship $\text{Prob}(response\ time \leq x) \geq y$. We use $SLA(x, y)$ to denote such an SLA.

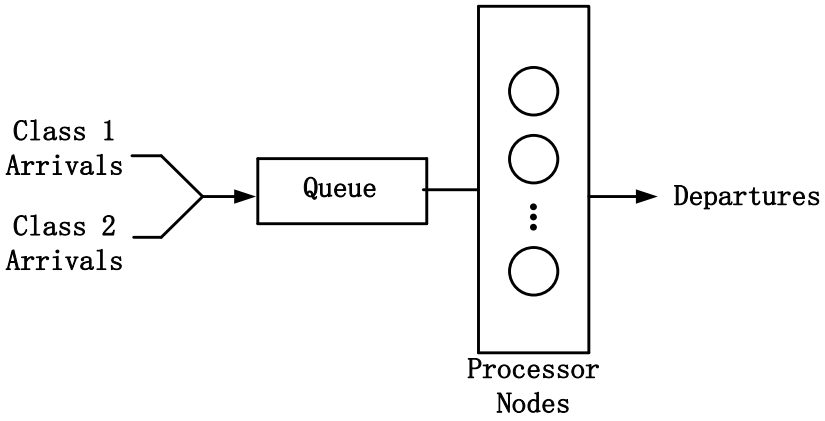

Figure 3.1: Shared allocation

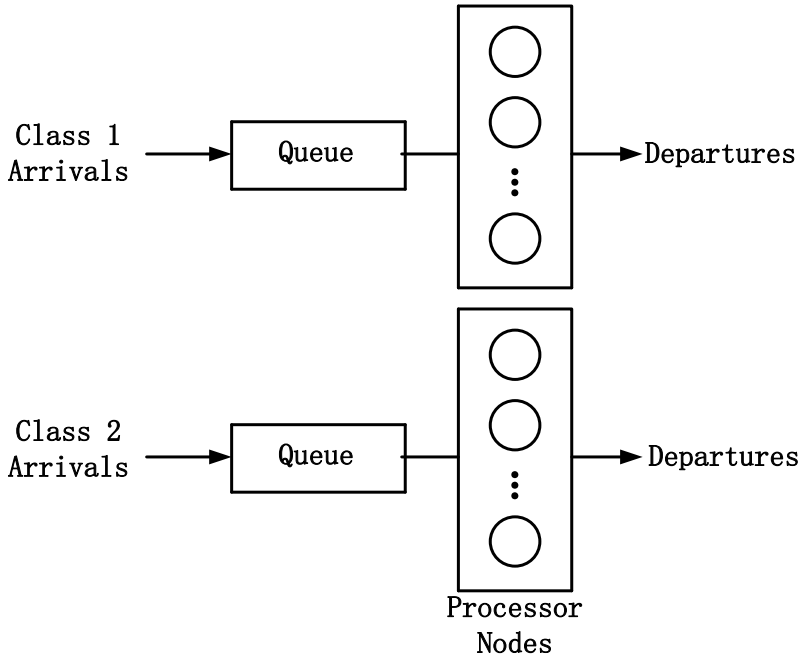

Figure 3.2: Dedicated allocation

In the remainder of this thesis, a comprehensive evaluation of SA and DA under FCFS over a range of workloads and SLAs will be performed in order to identify a strategy (SA or DA) that would result in the smaller number of processor nodes required

while meeting the SLAs of both classes; an investigation of the impact of scheduling disciplines is also included. Extension of our results to the case of three job classes will be investigated as well.

Chapter 4

Allocation strategies for two interactive classes

In this chapter, the performance of SA and DA is evaluated for the case of two interactive job classes. Our evaluation is based on the performance models shown in Figure 3.1 and 3.2. The scheduling discipline used for DA is FCFS because each job class is allocated a dedicated pool of processor nodes. However, for SA, a variety of scheduling disciplines can be considered to implement priority between job classes. We first investigate SA under FCFS which is the most common discipline. A comparative evaluation of DA and SA is presented in this chapter. We next investigate the impact of scheduling on performance by considering two priority disciplines, namely HOL, and a new discipline called PDP. Our results will be presented in Chapter 5.

4.1 Analytic results for FCFS

Under DA, the model for each job class can be viewed as an M/M/m model with FCFS scheduling. The same model is also applicable when FCFS is used for SA. For this model,

analytic results for the response time distribution are available in [22]. Let $F(x)$ be the cumulative distribution function (CDF) of response time of class i, i.e., $F(x) = \text{Prob}\,(response\ time \leq x)$, for $i = 1,2$. In [22], it was shown that:

$$F(x) = \begin{cases} 1 - e^{-\mu x} - P(0)\frac{m\rho^m \mu e^{-\mu x} x}{m!(m-\rho)} & if \quad \rho = m-1 \\ \sum_{n=0}^{m-1} P(0)(1-e^{-\mu x})\frac{\rho^n}{n!} \quad + & otherwise \\ P(0)\frac{m\rho^m}{m!(1-m+\rho)}\left(\frac{1-e^{-(m-\rho)\mu x}}{m-\rho} - 1 + e^{-\mu x}\right) & \end{cases} \tag{4.1}$$

where $P(0) = (\sum_{n=0}^{m-1}\frac{\rho^n}{n!} + \frac{m\rho^m}{m!(m-\rho)})^{-1}$ is the probability that the system is empty, $\rho = \lambda/\mu$ is the traffic intensity, and λ and μ are the arrival rate and service rate, respectively. Note that $m > \rho$, otherwise the system does not have sufficient capacity to handle the load.

4.2 Allocation strategies

Consider first DA. The results in Equation (4.1) can be used to determine m_{D1} and m_{D2}, the smallest number of processor nodes required to achieve the SLAs for class 1 and class 2, respectively. We observe that for an $SLA(x, y)$, it is required that $y < 1 - e^{-\mu x}$ because the response time cannot be smaller than the service time.

An algorithm that determines the smallest number of processor nodes is included in Algorithm 1. This algorithm starts with $m = \lfloor \rho \rfloor + 1$ and increases m until the target probability y is achieved. Let SLA_i be the SLA of class i, $i = 1,2$. m_{Di} can be

obtained by setting the arrival rate to λ_i, the service rate to μ, and $SLA(x, y)$ to SLA_i. Let m_D be the minimal number of processor nodes required under DA to meet the SLAs of both classes. m_D is given by:

$$m_D = m_{D1} + m_{D2} \tag{4.2}$$

Algorithm 1:

Input:	σ	//	Arrival rate
	μ	//	Service rate
	$SLA(x, y)$	//	Service level agreement (SLA)
Output:	m	//	Smallest number of processor nodes required
		//	such that SLA is met

1: $m = \lfloor \sigma/\mu \rfloor + 1$
2: while $(F(x) < y)\{\ m = m + 1\ \}$
3: return m

Consider next SA. Under FCFS, analytic results for the response time distribution can be obtained by extending the results in [23] to the case of multiple processor nodes. The resulting CDF is the same as that for the M/M/m – FCFS model with arrival rate equals to $\lambda = \lambda_1 + \lambda_2$. Furthermore, both classes have the same response time distribution. Let m_{si} be the number of processor nodes required under SA to meet SLA_i, $i = 1,2$. m_{si} can be obtained from Algorithm 1 by setting the arrival rate to $\lambda_1 + \lambda_2$, the service rate to μ, and $SLA(x, y)$ to SLA_i. m_S, the smallest number of processor nodes required to meet the SLAs of both classes is then given by:

$$m_S = \max(m_{S1}, m_{S2}) \tag{4.3}$$

4.3 DA and SA comparison

In this section, we use numerical examples to evaluate the performance difference of DA and SA under FCFS scheduling. The input parameters considered are shown in Table 4.1, where λ_i is the arrival rate of class i, and x_i and y_i are parameters of SLA_i, representing the response time threshold and target probability, respectively. We restrict the values of λ_1 and λ_2 such that $\lambda_1 + \lambda_2 \leq K = 40$. We feel that this represents a sufficiently wide range of workload. The service rate μ is set to 1.

λ_i	0.1, 0.2, … , 40.0
x_i	2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0
y_i	80%, 85%, 90%, 95%

Table 4.1: Parameter values of arrival rates and SLAs

Our evaluation is based on the total number of processor nodes required to meet the SLA of both classes, as given by m_D and m_S in Equations (4.2) and (4.3), respectively. For each combination of λ_i, x_i and y_i $(i = 1,2)$, SA (or DA) is superior if $m_s < m_D$ (or $m_D < m_S$).

With 7 values for x_i and 4 values for y_i, there are 28 possible SLAs for each class. However, 3 of them are not used because the condition $y < 1 - e^{-\mu x}$ is not met. They are $x_i = 2.0$ and $y_i = 95\%$; $x_i = 2.0$ and $y_i = 90\%$; and $x_i = 2.5$ and $y_i = 95\%$.

Our results show that when both classes have the same SLA, SA always performs better than, or has the same performance as, DA. However, when SLA_1 and SLA_2 are

different, neither SA nor DA is superior for all combinations of parameter values. For example, the results for two selected cases, shown in Table 4.2, indicate that DA is superior for case 1, but SA is superior for case 2. Furthermore, we are not able to come up with simple rules to identify the preferred strategy (DA or SA). This is due to the large number of combinations of λ_i, x_i and y_i $(i = 1,2)$ that need to be considered. A heuristic algorithm to determine the preferred strategy will be presented in the next section.

	λ_1	SLA_1	λ_2	SLA_2	m_D	m_S
Case 1	0.6	SLA(3, 0.80)	3.0	SLA(5, 0.95)	5	6
Case 2	0.6	SLA(3, 0.95)	3.6	SLA(5, 0.80)	8	7

Table 4.2: Two example cases

4.4 Heuristic algorithm for two interactive classes

In this section, we develop a heuristic algorithm that determines the preferred strategy (DA or SA) based on arrival rates and SLAs of the two job classes. Once the strategy is known, Algorithm 1 and Equations (4.2) and (4.3) can be used to obtain the number of nodes required.

4.4.1 SLA difference

To develop our heuristic algorithm, we first reduce the number of combinations involving

SLA_1 and SLA_2 by defining a measure that would characterize their difference. We note that for a given SLA, different arrival rates could result in different number of processor nodes required (denoted by m). In Figure 4.1, we plot the value of m against the arrival rate λ for two example SLAs. We observe that the value of m for SLA_1 is always larger than or equal to that for SLA_2.

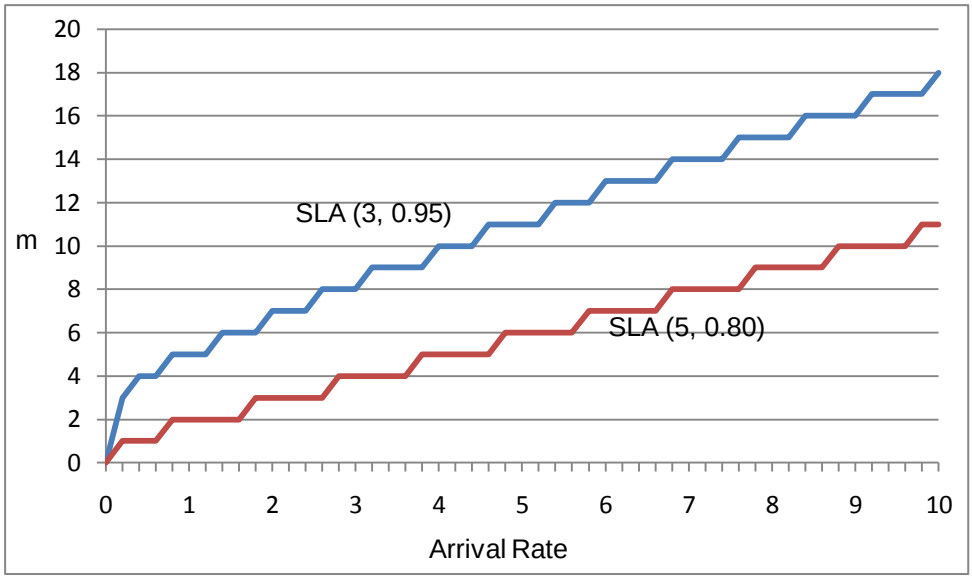

Figure 4.1: Number of processor nodes required

Through extensive testing, the following pattern is observed. Let $m(\lambda, SLA)$ be the smallest number of processor nodes required for the given λ and SLA. For any pair of SLAs (SLA_1 and SLA_2), either

$$m(\lambda_1, SLA_1) \geq m(\lambda_2, SLA_2)$$

or $$m(\lambda_1, SLA_1) \leq m(\lambda_2, SLA_2)$$

for all values of λ under consideration (which is $0 < \lambda \leq 40$). This pattern leads us to

use a single metric to describe the difference in m for a pair of SLAs.

Let $G(SLA)$ be the average number of processor nodes required to meet the given SLA over the range of arrival rates considered. $G(SLA)$ is given by:

$$G(SLA) = \frac{1}{K}\int_0^K m(x, SLA)dx \tag{4.4}$$

where $K = 40$, the upper limit of the range of λ under consideration. We define a metric called "SLA Difference" between SLA_1 and SLA_2 (denoted by D) as follows:

$$D = G(SLA_1) - G(SLA_2) \tag{4.5}$$

We now present results that show the impact of D on the merits of SA and DA. Consider the two scenarios summarized in Table 4.3. The SLA pair for scenario 1 is not the same as that for scenario 2, but the SLA differences of the two scenarios are almost the same (equal to 22.6). The results for these two scenarios are shown in Figures 4.2 and 4.3, respectively. For each combination of λ_1 and λ_2, the corresponding intersection is marked red if DA is the better strategy, and green if SA is better than or as good as DA. We observe similar patterns for both scenarios 1 and 2. Let f_D be the fraction of intersections that are red (i.e., DA is better). Our results indicate that for both scenarios, f_D is approximately 5.2%. The same observation is made from the results in Figures 4.4 and 4.5 where we consider two other scenarios that have larger SLA differences (see Table 4.4). For these scenarios, SLA difference D is 83.45 and the resulting f_D is increased to about 64%.

	SLA_1	SLA_2	D	f_D
Scenario 1	SLA(4.5, 0.85)	SLA(2.5, 0.90)	22.58	5.21%
Scenario 2	SLA(4.0, 0.80)	SLA(2.5, 0.90)	22.60	5.21%

Table 4.3: SLA pairs where $D = 22.6$

	SLA_1	SLA_2	D	f_D
Scenario 3	SLA(5.0, 0.85)	SLA(3.0, 0.95)	83.45	63.98%
Scenario 4	SLA(4.5, 0.80)	SLA(3.0, 0.95)	83.48	64.04%

Table 4.4: SLA pairs where $D = 83.5$

Through extensive testing, it was found that the same observation is true for other scenarios where the SLA differences are close to each other, namely,

1. The combinations of λ_1 and λ_2 where DA is better are almost identical.
2. The values of f_D are very similar.

We also observe that f_D tends to increase with SLA difference. This is illustrated in Figure 4.6 where f_D is plotted against the SLA difference. From the results in Figures 4.2 to 4.6, we conclude that the SLA difference D is potentially useful in our effort to develop a heuristic algorithm that determines whether DA or SA is a preferred strategy. This issue will be addressed in the next subsection.

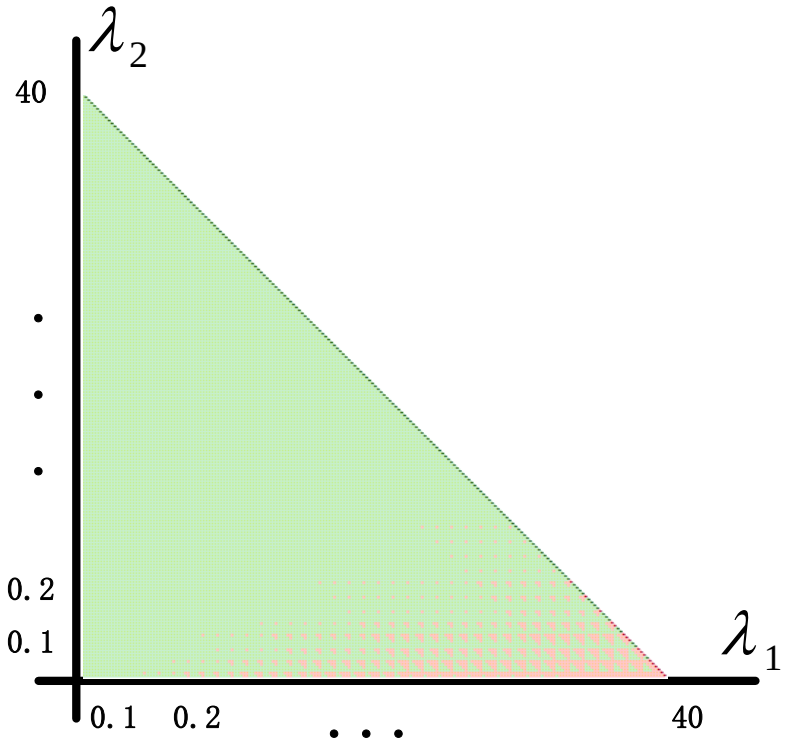

Figure 4.2: Scenario 1

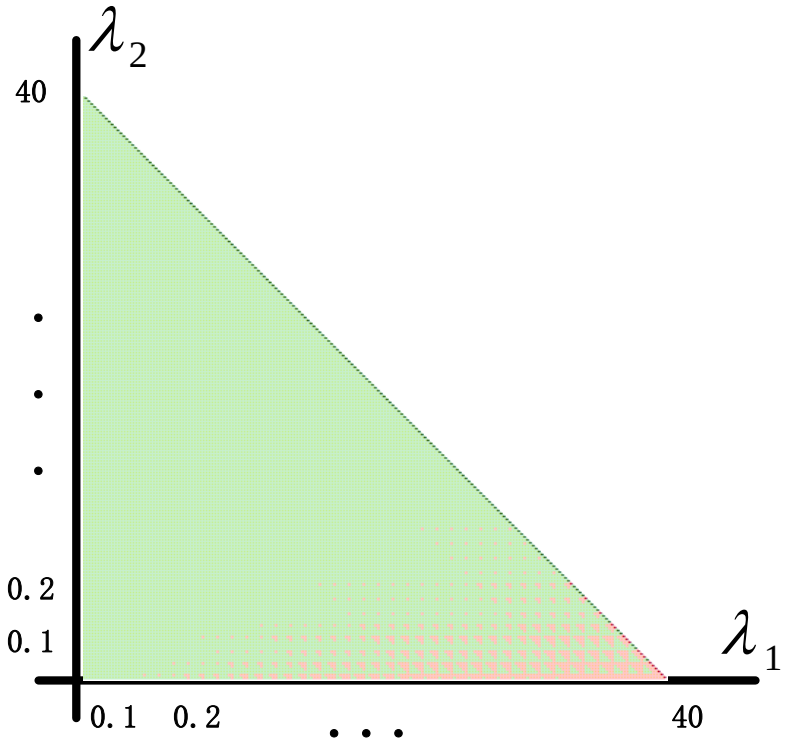

Figure 4.3: Scenario 2

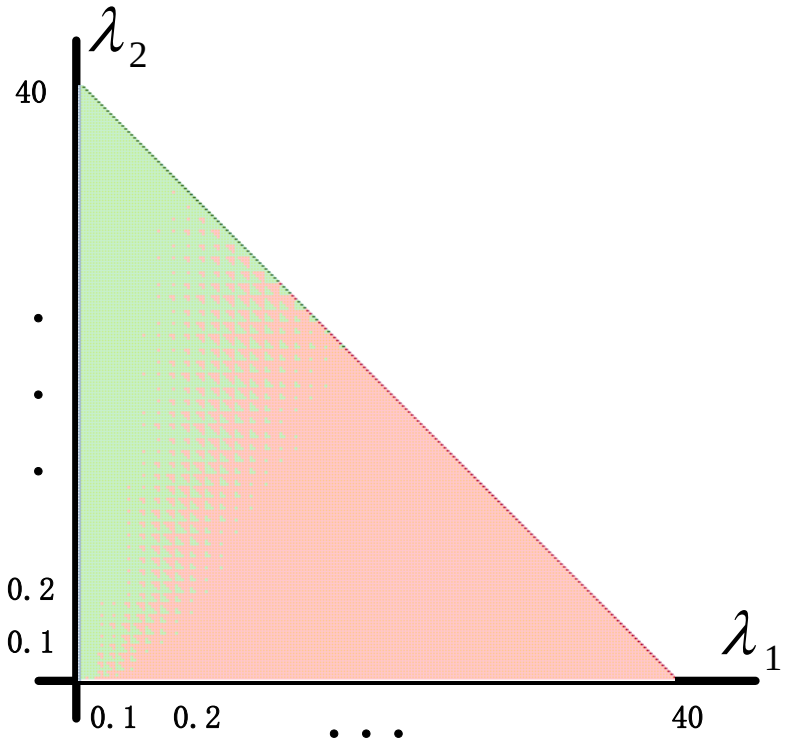

Figure 4.4: Scenario 3

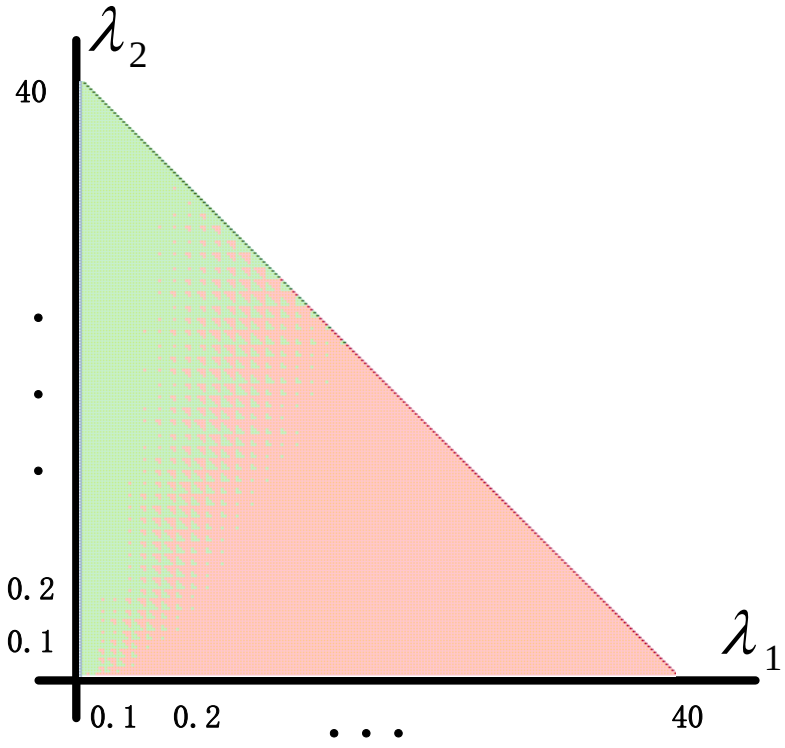

Figure 4.5: Scenario 4

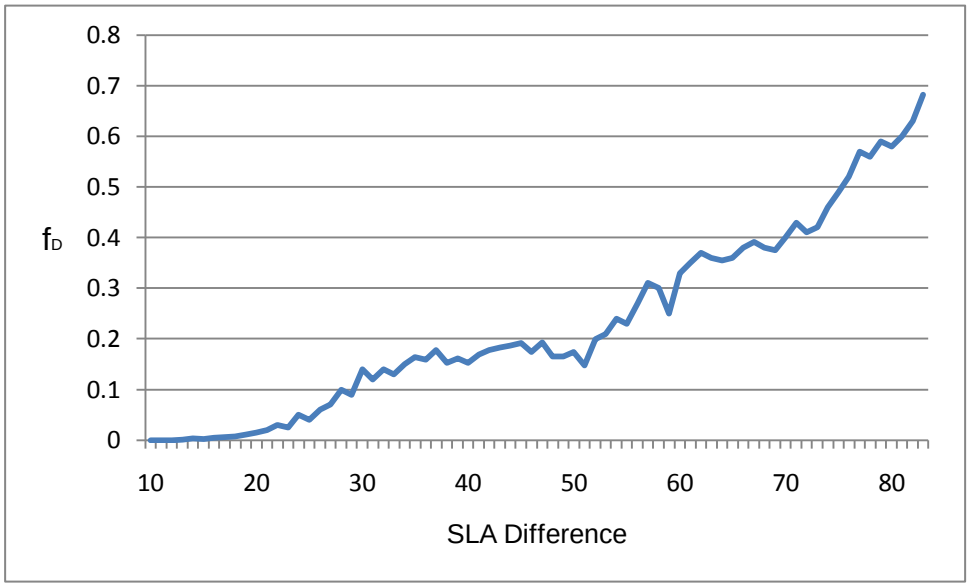

Figure 4.6: Percentage of DA vs. SLA difference

4.4.2 Heuristic algorithm

Our heuristic algorithm is based on the observation that there are well-defined regions in Figures 4.2 to 4.5 (and numerical examples for other values of SLA difference) where DA or SA is very likely to be the preferred strategy. These regions are separated approximately by a straight line, as illustrated in Figure 4.7. We thus define, for a given SLA difference, an angle α such that at least $q\%$ of intersections in region 2 indicate that DA is the preferred strategy. In our investigation, we use $q = 90$. Using numerical examples, a plot of the angle α against SLA difference is shown in Figure 4.8. We observe that the angle α tends to increase with SLA difference.

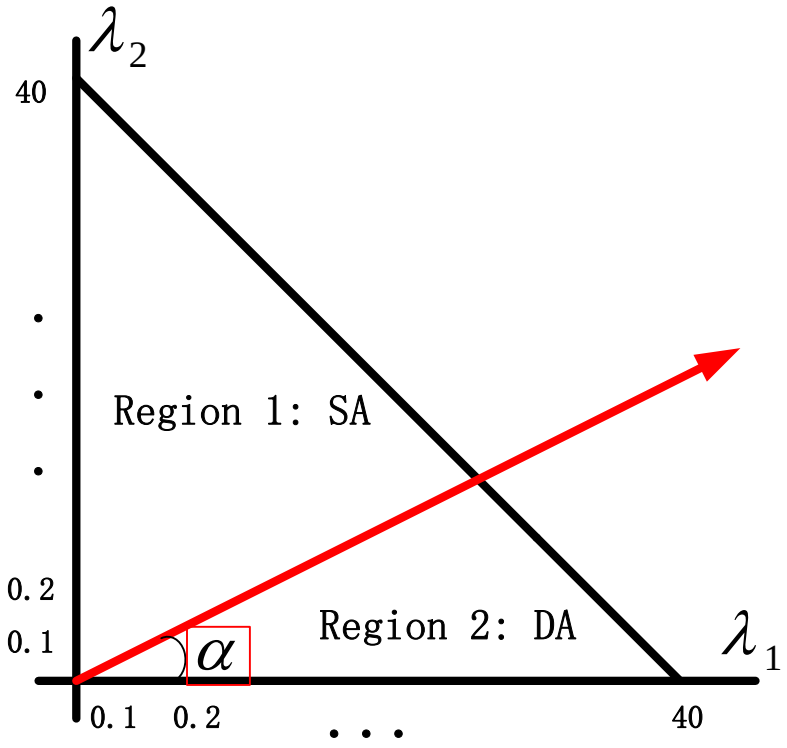

Figure 4.7: Heuristic method

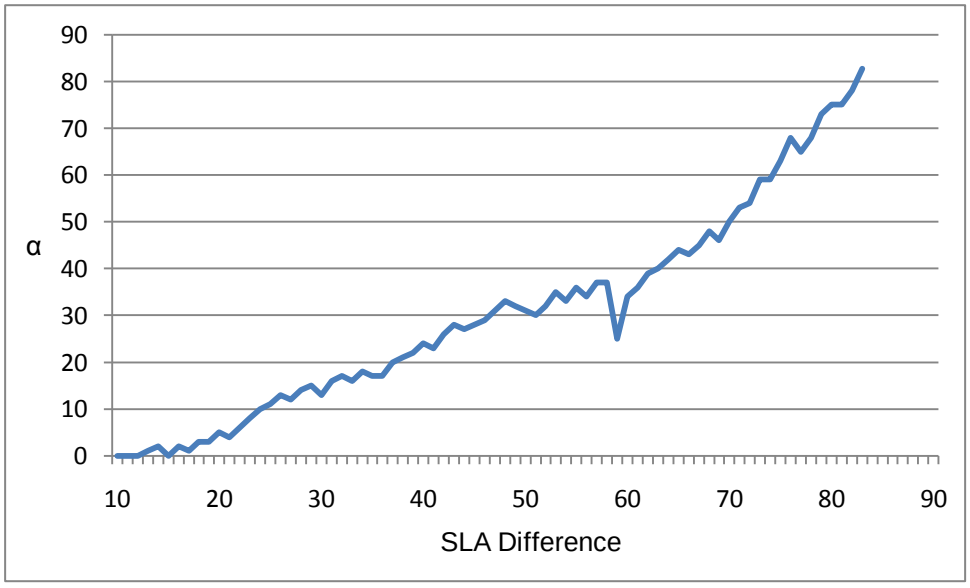

Figure 4.8: Angle vs. SLA difference

In our algorithm, we use an "angle table" which relates the angle α to a given SLA difference. An example of such a table is shown in Table 4.5 where the SLA difference is organized into 5 intervals. An angle α is pre-determined for each interval; the pre-determined value is the average of the α's for the SLA differences within the interval.

SLA Difference	Angle α (degree)
[0, 30)	0
[30, 62)	22
[62, 78)	52
[78, 82)	69
[82, 86.1)	77

Table 4.5: Angle table

Our algorithm is included as Algorithm 2. It operates as follows. We first compute $G(SLA_1)$ and $G(SLA_2)$ using Equation (4.4). These values are then used to compute the SLA difference D. The angle α corresponding to D is obtained from the angle table. The values of λ_1, λ_2 and α, are then used to identify the preferred strategy. Specifically, if the intersection (λ_1, λ_2) is below the line defined by the angle α (i.e., in region 2 of Figure 4.7), DA is the preferred strategy; otherwise SA is the preferred strategy.

Algorithm 2:

Input:	λ_1, λ_2	//	Arrival rates
	SLA_1, SLA_2	//	SLAs
Output:	DA or SA	//	Allocation Strategy

1: Compute $G(SLA_1)$ and $G(SLA_2)$
2: Compute SLA difference $G(SLA_1) - G(SLA_2)$
3: Search angle table to obtain the value of α
4: if ($\tan^{-1}\frac{\lambda_2}{\lambda_1} \leq \alpha$) { return DA }
5: else { return SA }

4.5 Performance evaluation

In this section, the heuristic algorithm presented in Section 4.4 is evaluated with respect to its ability to come up with a strategy (DA or SA) that results in the smallest number of processor nodes required.

4.5.1 Methodology

Our evaluation is based on the following consideration. Each time the global arbiter makes resource allocation decisions, it determines the number of processor nodes required by the two job classes, using as input parameters such as λ_i, x_i and y_i $(i = 1, 2)$. Since these parameters may have different values at different time instants when resource allocation decisions are made, our approach is to consider a large number L, of combinations of λ_i, x_i and y_i $(i = 1, 2)$. The performance of the heuristic

algorithm for each combination is determined, and the average performance over the L combinations is used for evaluation purposes.

For each combination, the values of λ_i, x_i and y_i $(i = 1, 2)$ are selected according to their respective probability distributions. These values are generated using random numbers. The probability distributions used in our evaluation are summarized in Table 4.6. Note that for λ_1 and λ_2, three different distribution are considered, representing different frequencies of values of λ_1 and λ_2 seen by the global arbiter at decision points. Only one distribution is used for each of the other parameters. The notation used in Table 4.6 is explained as follows:

- U (a, b) – uniform between a and b
- N (20, σ^2) – normal with mean 20 and variance σ^2 (values $\leq$ 0 and > 40 are excluded)
- E (t) – exponential with mean t (values > 40 are excluded)

Parameter	Distribution
λ_1, λ_2	U (0, 40)
	N (20, σ^2)
	E (t)
x_1, x_2	U (a, b)
y_1, y_2	U (0.80, 0.95)

Table 4.6: Probability distributions

4.5.2 Performance results

For our heuristic algorithm, its effectiveness is measured by: $S = \text{Prob}(heuristic\ algorithm\ finds\ a\ correct\ strategy)$. By correct strategy, we mean a strategy that results in the smallest number of nodes required to meet the SLAs of both job classes. In case both DA and SA lead to the same smallest number, then either strategy can be considered a correct strategy. The performance metric S is obtained as follows. We repeat the steps shown in Procedure 1 L times (the initial value of the variable $n_{correct}$ is zero). S is then given by: $S = n_{correct}/L$. Our results for $L = 10{,}000$ and 6 different settings of the parameter values are shown in Table 4.7. For all settings, the distribution used for y_1 and y_2 is U (0.8, 0.95).

Procedure 1:

1: Generate values for λ_1 and λ_2.
2: if $\lambda_1 + \lambda_2 > 40$, then goto step 1.
3: Generate values for x_1, x_2, y_1 and y_2.
4: Apply Algorithm 2 to obtain an allocation strategy (denoted by R_1).
5: Compute m_D and m_S given by Equation (4.2) and (4.3), respectively. The correct strategy (denoted by R_2) is DA if $m_D \leq m_S$ or SA if $m_S \leq m_D$.
6: if R_1 is the same as R_2, then $n_{correct}$ ++.

The results in Table 4.7 show that our heuristic algorithm has at least a 96% probability of finding a correct strategy for all the cases considered. These results indicate that the heuristic algorithm is effective in determining a correct strategy.

$\lambda_1,\ \lambda_2$	$x_1,\ x_2$	S
U (0, 40)	U (2, 5)	0.973
U (0, 40)	U (2, 10)	0.979
N (20, 5)	U (2, 5)	0.961
N (20, 10)	U (2, 5)	0.966
E (10)	U (2, 5)	0.982
E (20)	U (2, 5)	0.984

Table 4.7: Probability of correct strategy: Algorithm 2

Chapter 5

Priority disciplines

In this chapter, we consider scenarios where the queueing discipline is not restricted to FCFS. Obvious choices are disciplines that give priority to the job class that has a more demanding SLA, e.g., a smaller response time threshold x and/or a larger target probability y. Such disciplines are only applicable under SA. Two priority disciplines are considered: head of the line priority (HOL) [22] and a new discipline called probability dependent priority (PDP). The performance difference of HOL, PDP and FCFS is evaluated based on the number of processor nodes required to meet the SLAs of both job classes.

5.1 Head of the line priority

Under head of the line priority (HOL), jobs belong to different priority classes. The job class with the larger $G(SLA)$ value has higher priority. Whenever a processor node becomes available, jobs in the higher priority class are considered first. If the queue of the higher priority class is empty, then jobs in the lower priority class are considered. Within

the same class, jobs are served in FCFS order.

5.2 Probability dependent priority

Probability dependent priority (PDP) is a new queueing discipline designed to maximize the probability of meeting a given response time goal. This should have a positive effect in terms of minimizing the number of processor nodes required to meet the SLAs of both classes. Let τ_i be the measured frequency that response time $\leq$ the threshold x_i. The following two counters are used in PDP (both are zero initially):

- $total_i$ – number of class i jobs completed so far
- met_i – number of completed class i jobs that has response time $\leq x_i$

Each time a class i job completes service, $total_i$ is incremented by one; if this job has response time $\leq x_i$, met_i is also incremented by one. τ_i is then given by: $\tau_i = met_i/total_i$. The priority of class i, defined in Equation (5.1) below, is updated.

$$P_i = y_i - \tau_i \qquad i = 1,2 \tag{5.1}$$

In PDP, the job class with the larger P_i has higher priority. Whenever a processor node becomes available, jobs in the higher priority class are considered first. If the queue of the higher priority class is empty, then jobs in the lower priority class are considered. Within the same class, jobs are served in FCFS order. In case both classes have the same priority value, then the next job class to receive service is selected at random.

Note that with PDP, the job class that is meeting the SLA with the smaller margin is

given higher priority. Note also that the priority of a job class may change over time because τ_i is updated each time a class i job completes service.

5.3 Performance evaluation

In this section, the performance difference of the two priority scheduling disciplines (HOL and PDP) and FCFS is investigated. For FCFS, results are provided by the heuristic algorithm in Section 4.4. As to HOL and PDP, analytic results for the response time distribution are difficult to obtain, so simulation is used. In order to get reliable steady state results, we perform 100 experiments with selected parameters and determine a length of simulation run using the criterion that with 10 replications, the width of the 95% confidence interval of the mean number of jobs in system is within $\pm 5\%$ of the sample mean. Our results show that the above criterion is met with a length of run of 20,000 time units.

Let m_F, m_H and m_P be the smallest number of processor nodes required by FCFS, HOL and PDP, respectively, such that the SLAs of both classes are met. We say that

- FCFS is a top discipline if $m_F \leq m_H$ and $m_F \leq m_P$;
- HOL is a top discipline if $m_H \leq m_F$ and $m_H \leq m_P$; and
- PDP is a top discipline if $m_P \leq m_F$ and $m_P \leq m_H$.

The methodology presented in Section 4.5.1 is used in our evaluation. The performance metrics are q_F, q_H and q_P, the fraction of times that FCFS, HOL, and PDP are a top

discipline, respectively, among the L combinations of parameter values considered. The steps shown in Procedure 2 are repeated L times (the initial values of the variable n_F, n_H and n_P are zero). q_F, q_H and q_P are then given by $q_F = n_F/L$, $q_H = n_H/L$, and $q_P = n_P/L$, respectively.

Procedure 2:

1: Generate values for λ_1 and λ_2.
2: if $\lambda_1 + \lambda_2 > 40$, then goto step 1.
3: Generate values for x_1, x_2, y_1 and y_2.
4: Apply Algorithm 2 to obtain a correct strategy for FCFS and use Equation (4.2) and (4.3) to determine m_F.
5: Obtain m_H and m_P by simulation.
6: if $m_F \leq m_H$ and $m_F \leq m_P$, then n_F++.
7: if $m_H \leq m_F$ and $m_H \leq m_P$, then n_H++.
8: if $m_P \leq m_F$ and $m_P \leq m_H$, then n_P++.
9: if $m_P < m_F$ and $m_P < m_H$, then $n++$, $s_F += m_F - m_P$, $s_H += m_H - m_P$.

Our results for $L = 10{,}000$ and 6 different settings of the probability distributions are shown in Table 5.1. These results show that PDP is superior to HOL and FCFS in terms of the fraction of time that it is a top discipline. Specifically, PDP is a top discipline over 97% of the time, compared to less than 30% for HOL and less than 2% for FCFS.

To further characterize the advantage of PDP, we compute the average difference in number of processor nodes required between PDP and the other disciplines among those combinations of λ_i, x_i and y_i $(i = 1,2)$ where PDP is the top discipline (i.e., $m_P < m_F$ and $m_P < m_H$). This is done by step 9 in Procedure 2 where s_F and s_H are used to accumulate the difference between PDP and FCFS and between PDP and HOL,

respectively; and n is used to keep track of the number of combinations where PDP is the top discipline (n, s_F and s_H are initially 0). The average differences are then given by $\Delta_F = s_F/n$ and $\Delta_H = s_H/n$, respectively. Results for Δ_F and Δ_H for the 6 settings of the probability distributions are shown in Table 5.2. These results show that the reduction in number of processor nodes required is consistent across probability distributions, with an average of 1.41 compared to FCFS and 0.45 compared to HOL.

$\lambda_1,\ \lambda_2$	$x_1,\ x_2$	q_F	q_H	q_P
U (0, 40)	*U* (2, 5)	1.6%	25.5%	98.3%
U (0, 40)	*U* (2, 10)	0.9%	29.4%	97.1%
N (20, 5)	*U* (2, 5)	1.3%	24.8%	98.5%
N (20, 10)	*U* (2, 5)	1.1%	23.1%	98.8%
E (10)	*U* (2, 5)	1.4%	27.5%	98.4%
E (20)	*U* (2, 5)	1.5%	24.6%	98.0%

Table 5.1: Performance comparison: 2 job classes

$\lambda_1,\ \lambda_2$	$x_1,\ x_2$	Δ_F	Δ_H
U (0, 40)	*U* (2, 5)	1.41	1.29
U (0, 40)	*U* (2, 10)	1.36	1.16
N (20, 5)	*U* (2, 5)	1.49	1.29
N (20, 10)	*U* (2, 5)	1.43	1.26
E (10)	*U* (2, 5)	1.33	1.12
E (20)	*U* (2, 5)	1.36	1.17

Table 5.2: Performance difference: 2 job classes

Chapter 6

Allocation strategies for three interactive classes

In Chapter 4, we investigated resource allocation strategies for two interactive job classes. For the case of FCFS scheduling, there are two allocation strategies: DA and SA. When there are $N > 2$ job classes, the number of allocation strategies increases quickly with N, and the problem of determining the preferred strategy becomes quite complicated. This can be illustrated by examining an example scenario of three job classes.

When there are three job classes, the number of AEs could be 1, 2, or 3. With one AE, the allocation strategy is SA, i.e., all three classes share a pool of processor nodes. For the case of 2 AEs, we have a mixed scenario where two of the job classes are in the same AE (under SA) while the remaining job class has its own allocation (under DA). There are 3 allocation strategies depending on which of the two classes are in the same AE. Finally, when there are 3 AEs, the allocation strategy is DA, i.e., each job class has its own dedicated resources. Thus, there are five possible allocation strategies. Furthermore, the number of combinations of arrival rates and SLAs that need to be considered is larger

when compared with that of two job classes.

Based on the above discussions, we see that the problem becomes more complex when $N > 3$ job classes. In this chapter, we consider the case of $N = 3$ and develop a heuristic algorithm for resource allocation. This would provide further insight into the performance of different allocation strategies with a modest increase in complexity. The approach used is similar to that used in Chapter 4, namely, we analyze the performance of different allocation strategies using numerical examples and use the results to come up with a heuristic algorithm that determines the preferred strategy for given values of arrival rates and SLAs of the three job classes.

6.1 Allocation strategies

In this section, we define the five resource allocation strategies and the notation that will be used in subsequent discussions. Let λ_i and SLA_i be the arrival rate and SLA of class i, ($i = 1, 2$ and 3). The five strategies are:

- DA - each job class is in a separate AE. Let m_{Di} be the number of nodes needed for class i to meet its SLA ($i = 1, 2$ and 3). m_{Di} can be obtained by using Algorithm 1 with the arrival rate set to λ_i, the service rate to μ and $SLA(x, y)$ to SLA_i. m_D, the smallest number of nodes required under DA is given by:

$$m_D = m_{D1} + m_{D2} + m_{D3} \tag{6.1}$$

- SA - all three job classes are in the same AE. We use m_S to denote the smallest

number of processors required under SA. The value of m_S is affected by the choice of scheduling discipline.

- Mixed – two of the three classes, say classes j and k, are in the same AE and the remaining class, say class l, is in a separate AE. There are three mixed strategies, one for each of the three combinations of values of j, k and l. Let $m_{S_{jk}}$ be the smallest number of nodes required to meet SLA_j and SLA_k under SA, and m_{D_l} be the smallest number of nodes required to meet SLA_l under DA. m_{D_l} can be obtained by using Algorithm 1 with the arrival rate set to λ_l, the service rate to μ and $SLA(x, y)$ to SLA_l. The value of $m_{S_{jk}}$ is affected by the choice of scheduling discipline under SA. The smallest number of nodes required to meet all SLAs, denoted by $m_{S_{jk}\,D_l}$, is then given by:

$$m_{S_{jk}\,D_l} = m_{S_{jk}} + m_{D_l} \tag{6.2}$$

6.2 Heuristic algorithm for three interactive classes

In this section, we develop a heuristic algorithm for case of FCFS scheduling. We first determine the value of m_S under SA. Let m_{Si} be the smallest number of nodes required to meet SLA_i ($i = 1, 2$ and 3). m_{Si} can be obtained by using Algorithm 1 with the arrival rate set to $\lambda_1 + \lambda_2 + \lambda_3$, the service rate to μ and $SLA(x, y)$ to SLA_i. m_S is then given by:

$$m_S = \max(m_{S1}, m_{S2}, m_{S3}) \tag{6.3}$$

A similar result can be derived for $m_{S_{jk}}$ under mixed strategy. Let m_{Sj} and m_{Sk} be the smallest number of nodes required to meet SLA_j and SLA_k, respectively, under SA. m_{Sj} is obtained by using Algorithm 1 with the arrival rate set to $\lambda_j + \lambda_k$, the service rate to μ and $SLA(x, y)$ to SLA_j; similarly for m_{Sk}. $m_{S_{jk}}$ is then given by:

$$m_{S_{jk}} = \max(m_{Sj}, m_{Sk}) \tag{6.4}$$

Consider now the development of our heuristic algorithm. Suppose each of the three job classes is represented by a vertex in a graph, denoted by c_1, c_2 and c_3, respectively. We apply Algorithm 2 to each pair of job classes. Two vertexes are connected if and only if Algorithm 2 recommends SA for the two corresponding job classes. Figure 6.1 demonstrates four possible graph structures after applying Algorithm 2.

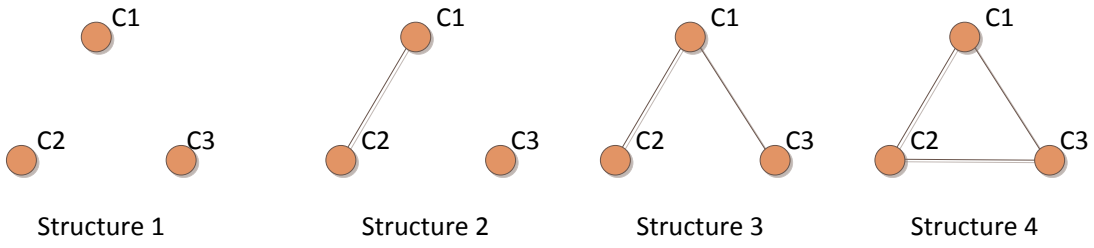

Figure 6.1: Resource allocation graph structures

The next question is how to translate these structures to allocation strategies. This is straightforward for structure 1, 2 and 4. Structures 1 and 4 correspond to DA and SA, respectively. Structure 2 is translated to a mixed strategy with SA for classes 1 and 2 and DA for class 3. However, the allocation strategy corresponding to structure 3 is not

unique because class 1 cannot be in two different AEs at the same time. For this structure, there are four possible solutions, as depicted in Figure 6.2:

1. Add an edge from c_2 to c_3, resulting in SA.
2. Remove the edge with small SLA difference, resulting in a mixed strategy with class 1 and class 2 in the same AE.
3. Remove the edge with large SLA difference, resulting in a mixed strategy with class 1 and class 3 in the same AE.
4. Randomly choose one of the above with equal probability.

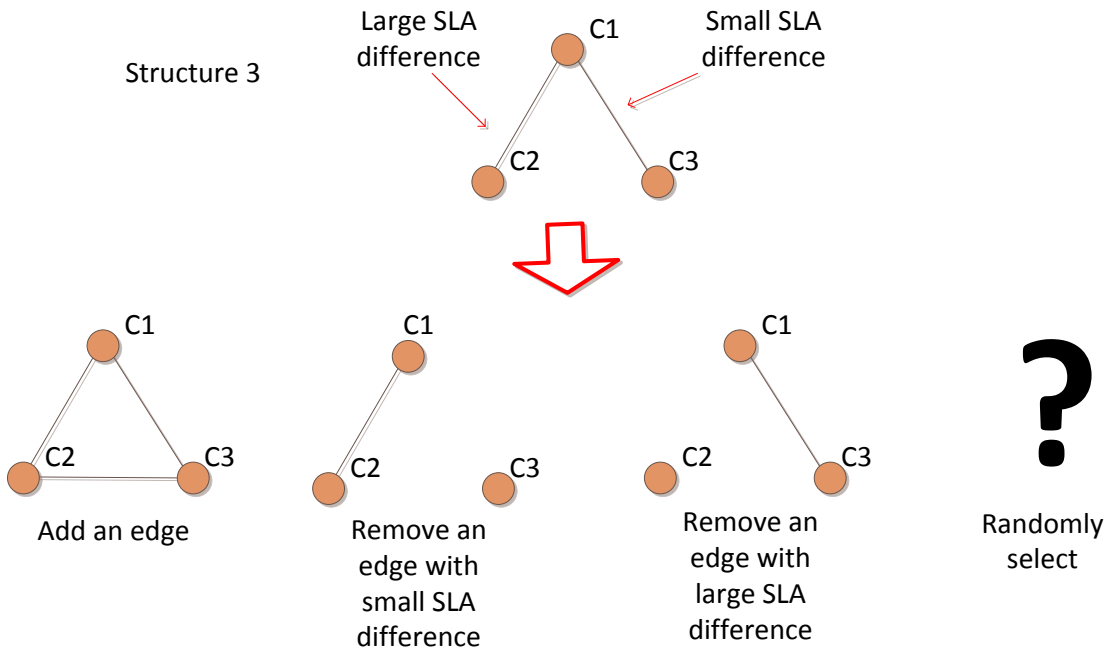

Figure 6.2: Solutions to allocation structure 3

We use the methodology presented in Section 4.5.1 to evaluate the merits of the four solutions mentioned above. For each solution, we determine the average number of

processor nodes allocated (denoted by $M_{avg,j}$ for solution j, $j = 1, 2, 3$ and 4). $M_{avg,j}$ is obtained by repeating the steps shown in Procedure 3 L times ($L = 10{,}000$). Note that step 7 is reached when a graph with structure 3 (as shown in Figure 6.1) is found. The initial value of the variable $m_{total,j}$ is zero, $j = 1, 2, 3$ and 4. $M_{avg,j}$ is then given by: $M_{avg,j} = m_{total,j}/L$.

Procedure 3:

1: Generate values for λ_1, λ_2 and λ_3.
2: if $\lambda_1 + \lambda_2 + \lambda_3 > 40$, then goto step 1.
3: Generate values for x_1, x_2, x_3, y_1, y_2 and y_3.
4: Run Algorithm 2 for each pair of classes.
5: Create resource allocation graph T.
6: if T is translated to a unique resource allocation strategy, then goto step 1.
7: Compute smallest number of nodes required for each of the four solutions (m_j is the smallest for solution j, $j = 1, 2, 3$ and 4).
8: $m_{total,1} += m_1$, $m_{total,2} += m_2$, $m_{total,3} += m_3$, and $m_{total,4} += m_4$.

The results are shown in Table 6.1. We observe that solution 3 has the smallest average number of nodes among the four solutions. This is consistent with our observation in Chapter 4 where a larger SLA difference means it is more likely to choose DA (see Figure 4.6).

Solutions	$M_{avg,j}$
1. Add an edge	39.01
2. Remove an edge with small SLA difference	38.20
3. Remove an edge with large SLA difference	38.09
4. Randomly select	38.46

Table 6.1: Performance of the four solutions

We now describe our heuristic algorithm. It is shown in Algorithm 3 below. Algorithm 2 is first applied to each pair of job classes to create a resource allocation graph T. If T does not translate to a unique allocation strategy, remove the edge with the larger SLA difference. Algorithm 3 then returns the resource allocation strategy according to *T*.

Algorithm 3:

Input:	λ_1, λ_2 and λ_3	//	Arrival rates
	SLA_1, SLA_2 and SLA_3	//	SLAs
Output:	DA, SA or Mixed	//	Allocation Strategy

1: run Algorithm 2 for each pair of classes
2: create resource allocation graph T
3: if (T does not translate to a unique strategy)
4: remove an edge with large SLA difference
5: return resource allocation strategy according to T

6.3 Performance evaluation

In this section, the heuristic algorithm presented in Section 6.2 is evaluated with respect to its ability to come up with a strategy that leads to the smallest number of processor nodes required.

Similar to our evaluation for two classes, the effectiveness of this algorithm is measured by $S = \text{Prob}(heuristic\ algorithm\ finds\ a\ correct\ strategy)$. By correct strategy, we mean a strategy that results in the smallest number of nodes required to meet the SLAs of all three job classes, among all five possible strategies. In case two or more

allocation strategies lead to the same smallest number, then any of these strategies can be considered as a correct strategy. The performance metric S is obtained as follows. We repeat the steps shown in Procedure 4 L times (the initial value of the variable $n_{correct}$ is zero). S is then given by: $S = n_{correct}\ /L$.

Procedure 4:

1: Generate values for λ_1, λ_2 and λ_3.
2: if $\lambda_1 + \lambda_2 + \lambda_3 > 40$, then goto step 1.
3: Generate values for $x_1,\ x_2,\ x_3,\ y_1, y_2$ and y_3.
4: Apply Algorithm 3 to obtain an allocation strategy and its corresponding number of processor nodes (denoted by m).
5: Compute m_D and m_S given by Equation (6.1) and (6.3), respectively. And determine $m_{S_{12}D_3}$, $m_{S_{13}D_2}$ and $m_{S_{23}D_1}$ using Equation (6.2).
6: if $m = \min(m_D, m_S, m_{S_{12}D_3}, m_{S_{13}D_2}, m_{S_{23}D_1})$, then $n_{correct}$ ++.

Our results for $L = 10{,}000$ and 6 different setting of the probability distributions are shown in Table 6.2. For all settings, the distribution used for $y_1,\ y_2$, and y_3 is U (0.8, 0.95). The results in Table 6.2 show that our heuristic algorithm has at least a 93% probability of finding a correct strategy over all the cases under consideration. These results indicate that the heuristic algorithm is effective in determining a correct strategy for three job classes.

$\lambda_1, \lambda_2, \lambda_3$	$x_1,\ x_2,\ x_3$	S
U (0, 40)	U (2, 5)	0.961
U (0, 40)	U (2, 10)	0.978
N (20, 5)	U (2, 5)	0.931
N (20, 10)	U (2, 5)	0.936
E (10)	U (2, 5)	0.973
E (20)	U (2, 5)	0.962

Table 6.2: Probability of correct strategy: Algorithm 3

6.4 Probability dependent priority scheduling

In this section, we consider the use of non-FCFS scheduling disciplines where two or more job classes are in the same AE and compare the performance of such disciplines to FCFS. Two non-FCFS disciplines, HOL and PDP, were investigated in Chapter 5 for the case of two job classes. The results showed that PDP is superior. So we will consider PDP only when we extend our investigation to the case of three classes.

With three job classes, PDP can be used in the SA strategy where all three job classes are in the same AE, or in the AE that has two job classes in any of three mixed strategies. As in Chapter 5, simulation is used to obtain performance results since analytic results for response time distribution of PDP are difficult to obtain. All four strategies where PDP can be used (SA and three mixed strategies) are simulated and the best result among these four strategies is used in the comparison. For FCFS, results provided by the heuristic algorithm in Section 6.1 are used.

Let m_F and m_P be the smallest number of processor nodes required by FCFS and PDP, respectively, such that the SLAs of all three classes are met. We say that

- FCFS is a top discipline if $m_F \leq m_P$; and
- PDP is a top discipline if $m_P \leq m_F$.

In our evaluation, the methodology presented in Section 4.5.1 is used. The steps shown in Procedure 5 are repeated L times (the initial values of the variable n_F and n_P are zero). Our performance metrics are q_F and q_P, the fraction of times that FCFS and PDP are a top discipline, respectively, among the L combinations of parameter values considered. q_F and q_P are given by: $q_F = n_F/L$ and $q_P = n_P/L$.

Procedure 5:

1: Generate values for λ_1, λ_2 and λ_3.
2: if $\lambda_1 + \lambda_2 + \lambda_3 > 40$, then goto step 1.
3: Generate values for $x_1,\ x_2,\ x_3,\ y_1, y_2$ and y_3.
4: Apply Algorithm 3 to obtain a correct strategy for FCFS and determine m_F, the number of processor nodes required for this strategy.
5: Obtain m_P by simulation.
6: if $m_F \leq m_P$, then n_F++.
7: if $m_P \leq m_F$, then n_P++.
8: if $m_P < m_F$, then $n++,\ s_F += m_F - m_P$.

Our results for $L = 10{,}000$ and 6 different settings of the probability distributions are shown in Table 6.3. These results again show that PDP is superior to FCFS in terms of the fraction of time that it is a top discipline. Our results also indicate that among the four strategies where PDP is used, SA with PDP in one AE always has the best performance.

To further characterize the advantage of PDP, we compute, among those combinations of λ_i, x_i and y_i ($i = 1, 2$ and 3) where PDP is the top discipline (i.e.,

$m_P < m_F$), the average difference between PDP and FCFS. This is done by step 8 in Procedure 5 where s_F is used to accumulate the difference between PDP and FCFS; and n is used to keep track of the number of combinations where PDP is the top discipline (n and s_F are initially 0). The average differences are then given by $\Delta_F = s_F/n$. Results for Δ_F for the 6 settings of the probability distributions are shown in Table 6.4. These results show that performance advantage of PDP for three job classes is more significant than that of the two classes (see results for two classes in Table 5.2).

$\lambda_1, \lambda_2, \lambda_3$	$x_1,\ x_2,\ x_3$	q_F	q_P
U (0, 40)	*U* (2, 5)	1.1%	99.7%
U (0, 40)	*U* (2, 10)	0.4%	100%
N (20, 5)	*U* (2, 5)	0.9%	99.7%
N (20, 10)	*U* (2, 5)	0.7%	99.8%
E (10)	*U* (2, 5)	0.3%	99.9%
E (20)	*U* (2, 5)	0.9%	99.9%

Table 6.3: Performance comparison: 3 job classes

$\lambda_1, \lambda_2, \lambda_3$	$x_1,\ x_2,\ x_3$	Δ_F
U (0, 40)	*U* (2, 5)	1.82
U (0, 40)	*U* (2, 10)	1.67
N (20, 5)	*U* (2, 5)	1.88
N (20, 10)	*U* (2, 5)	1.85
E (10)	*U* (2, 5)	1.90
E (20)	*U* (2, 5)	1.86

Table 6.4: Performance difference: 3 job classes

Chapter 7

Conclusion and future work

7.1 Conclusion

In this thesis, we investigate strategies for allocating processor nodes to a number of job classes such that the SLA of each class is met. Our focus is on interactive jobs where the SLA is based on response time distribution. For the case of two job classes, we have investigated two allocation strategies under FCFS scheduling; these strategies are shared allocation (SA) and dedicated allocation (DA), respectively. A heuristic algorithm which determines an allocation strategy (SA or DA) that results in the smallest number of processor nodes is developed. The performance of this algorithm is evaluated over a range of operating conditions and the results indicate that it is able to find a correct allocation strategy in at least 96% of the cases evaluated.

The performance of SA with non-FCFS scheduling is also investigated. We consider two priority disciplines: head of the line priority (HOL) and a new discipline called probability dependent priority (PDP). Simulation results show that PDP is more effective than HOL and FCFS in terms of the number of processor nodes required to meet the

SLAs of the two job classes.

Furthermore, we extend our heuristic algorithm for FCFS scheduling to the case of three job classes. The performance of this algorithm is also evaluated over a range of operating conditions and the results show that it is able to determine a correct strategy in over 93% of all cases considered. The performance of PDP for three job classes is evaluated by simulation and the results confirm that PDP is again superior to FCFS. Its advantage over FCFS is more significant when compared to the case of two classes.

Our contributions are summarized as below.

1. For FCFS scheduling, we have obtained results which show that SA is not always the better resource allocation strategy compared to DA with respect to response time distribution when SLA is taken into consideration.
2. We have developed and evaluated a heuristic algorithm for FCFS scheduling that determines a resource allocation strategy that results in the smallest number of processor nodes required for the case of two job classes, as well as an extension of this algorithm to three job classes.
3. We have developed a novel scheduling algorithm (called PDP) that is superior in performance when compared to FCFS and HOL.

7.2 Future Work

Directions for future work include the following.

1. Resource provisioning for more than three job classes. In general, it is very complex to find an optimal resource allocation strategy where each AE may have a potentially large number of applications. The general optimization problem is NP hard [1]. Developing of heuristic algorithms for more than three job classes is a future research problem. The insights gained from our investigation may provide useful guidelines for such algorithms.
2. Performance of scheduling disciplines for more than three job classes. We have already shown that PDP used with SA is an advantageous allocation strategy for two or three job classes when compared to other strategies. The question of whether PDP used with SA will have superior performance in general should be investigated. There is also the potential for developing scheduling disciplines other than PDP that have good performance in terms of requiring small number of processor nodes.
3. Interactive and batch applications. Dynamic resource provisioning for an AE with multiple interactive and batch applications is a challenging task. This is because batch jobs have very different resource requirements and performance goals from interactive jobs. An effective resource allocation strategy that benefits both interactive and batch applications would be an important contribution to dynamic resource provisioning.

References

[1]. W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In Proceedings of the 1st International Conference on Autonomic Computing, 2004.

[2]. L. Kleinrock. Queuing Systems Volume 2: Computer Applications. Wiley-Inerscience, New York, 1976

[3]. M. N. Bennani, and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In Proceedings of the 2nd International Conference on Autonomic Computing, 2005.

[4]. M. Woodside, T. Zheng, and M. Litoiu. Service system resource management based on a tracked layered performance model. In Proceedings of the 3rd International Conference on Autonomic Computing, 2006.

[5]. T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai, Tracking time-varying parameters in software systems with extended kalman filters. In Proceedings of CASCON, 2005.

[6]. Q. Zhang, L. Cherkasova, and E. Smirni, A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In Proceedings of the 4th International Conference on Autonomic Computing, 2007.

[7]. X. Wang, D. Lan, G. Wang, X. Fang, M. Ye, Y. Chen, and Q. Wang, Appliance-based autonomic provisioning framework for virtualized outsourcing data center. In Proceedings of the 4th International Conference on Autonomic Computing, 2007.

[8]. G. Tesauro, R. Das, W. E. Walsh, and J. O. Kephart. Utility-function-driven resource allocation in autonomic systems. In Proceedings of the 2nd International Conference on Autonomic Computing, 2005.

[9]. P. Padala, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem, and K. Shin. Adaptive control of virtualized resources in utility computing environments. In Proceedings of the European Conference on Computer Systems, 2007.

[10]. Y. Diao, J.L. Hellerstein, S. Parekh, R. Griffith, G.E. Kaiser, and D. Phung. A control theory foundation for self-managing computing systems. IEEE journal on selected areas in communications, 23(12):2213-2222, 2005.

[11]. C. Karamanolis, M. Karlsson, and X. Zhu. Designing controllable computer systems. In Proceedings of the USENIX Workshop on Hot Topics in Operating Systems, 2005.

[12]. X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D.Gmach, R. Gardner, T. Christian and L. Cherkasova. 1000 Islands: Integrated capacity and workload management for the next generation data center. In Proceedings of the 5th International Conference on Autonomic Computing, 2008.

[13]. J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. In Proceedings of the 3rd International Conference on Autonomic Computing, 2006.

[14]. S. Ghanbari, G. Soundararajan, J. Chen, and C. Amza. Adaptive learning of metric correlations for temperature-aware database provisioning. In Proceedings of the 4th International Conference on Autonomic Computing, 2007.

[15]. P. Shivam, S. Babu, and J. Chase. Learning application models for utility resource planning. In Proceedings of the 3rd International Conference on Autonomic Computing, 2006.

[16]. G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. IEEE Internet Computing, 11(1):22-30, 2007.

[17]. P. Thomas, D. Teneketzis, and J. K. MacKie-Mason. A market-based approach to optimal resource allocation in integrated-services connection-oriented networks. Operations Research, 50(4), 2004.

[18]. K. Coleman, J. Norris, G. Candea, and A. Fox. OnCall: Defeating spikes with a free-market server cluster. In Proceedings of the 1st International Conference on Autonomic Computing, 2004.

[19]. S. Lalis, C. Nikolaou, D. Papadakis, and M. Marazakis. Market-driven service allocation in a QoS-capable environment. In First International Conference on Information and Computation Economies, 1998.

[20]. Y. Lu, T. Abdelzaher, C. Lu, L. sha, and X. Liu, Feedback control with queuing-theoretic prediction for relative delay guarantees in web servers, Real-Time and Embedded Technology and Applications Symposium, Toronto, 2003.

[21]. L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, Queuing model based network server performance control. Real-Time Systems Symposium, 81-89, 2002.

[22]. M. Kwok. Performance Analysis of Distributed Virtual Environments. Phd Thesis, University of Waterloo, 2006

[23]. J.W. Wong and S.S. Lam. "Queueing Network Models of Packet-Switching Networks, Part I: Open Networks", Performance Evaluation, 1982, 9-21