Autonomic Resource Management for a Cluster that Executes Batch Jobs

by

Lik Gan Alex Sung

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, 2006

Abstract

Resource management of large scale clusters is traditionally done manually. Servers are usually over-provisioned to meet the peak demand of workload. It is widely known that manual provisioning is error-prone and inefficient. These problems can be addressed by the use of autonomic clusters that manage their own resources. In those clusters, server nodes are dynamically allocated based on the system performance goals. In this thesis, we develop heuristic algorithms for the dynamic provisioning of a cluster that executes batch jobs with a shared completion deadline.

External factors that may affect the decision to use servers during a certain time period are modeled as a time-varying cost function. The provisioning goal is ensure that all jobs are completed on time while minimizing the total cost of server usage. Five resource provisioning heuristic algorithms which adapt to changing workload are presented. The merit of these heuristics is evaluated by simulation. In our simulation, the job arrival rate is time-dependent which captures the typical job profile of a batch environment. Our results show that heuristics that take into consideration the cost function perform better than the others.

Acknowledgements

I am very grateful to work under the supervision of Prof. Johnny Wong, whose expert guidance and experiences helped me navigate through my research. His patience and insightful comments have facilitated the completion of this thesis. Thanks are given to Prof. Srinivasan Keshav and Prof. David Taylor for taking time to be the readers of this thesis.

I would like to thank Michael, Herman and Creamy for bringing pleasant and enjoyable moments to me during my study. Finally, I would like to thank my parents and family for their love, support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the decreasing in price but increasing processing power of commodity hardware, server clusters are widely used to execute computationally intensive jobs. In such clusters, the workload typically changes with time. Some of the servers may therefore have a low utilization level, even in the "busy hours". When this happens, servers are over-provisioned – keeping enough resources to provide the capacity for the occasional peak demand. Over-provisioning of servers is not economical due to under-utilization of resources. Manual dynamic resource allocation may help, but it has the drawback that tuning in resource allocation is usually done over long time intervals on the order of hours or days. Prompt reactions to unexpected changes in workload are lacking. Therefore, on-demand provisioning has received a lot of attention in recent years.

Many studies have been carried out to investigate the use of dynamic server allocation for clusters. A summary of these studies is provided in Chapter 2. These studies are mostly concerned with the processing of interactive jobs (e.g., webpage requests). In general, jobs to be processed may belong to different classes, e.g., interactive jobs and batch jobs. Due to the differences in workload and performance requirements, provisioning approaches designed for interactive clusters are not directly applicable for provisioning in a batch environment. In the relatively few studies that are related to dynamic server allocation of batch job clusters, the focus was on the allocation framework [8, 21] and on allocation between multi-institutional grid clusters [26]. Not much is known concerning dynamic server allocation within a single cluster.

For batch jobs, the resource provisioning goals are usually related to maximizing throughput and the fraction of jobs meeting their completion deadlines. In contrast, for interactive jobs, the provisioning goals are often related to maintaining a certain level of the response time performance and CPU utilization. Therefore, different performance metrics are used. In addition, batch jobs may not share processors, i.e., a single processor executes only one job at a time. Jobs are queued at the scheduler until there is a free processor. This is the default behaviour of popular batch job schedulers, e.g., Condor [10], PBS [22] and LSF [18].

Processor allocation incurs a cost which may be a function of time. Consider, for example, a data centre that hosts both web content and batch job processing services, where each service is hosted on its own cluster and servers can be migrated between clusters. For the web content hosting services, the data centre may need to pay a penalty if the response time percentile exceeds the requirements defined in the service level agreement. As is known, a web content service usually experiences a higher workload during the day and a lower workload after midnight. We may wish to discourage the allocation of servers to the batch job cluster in the daytime and allocate more servers to the web content cluster. Such discouragement can be expressed in terms of the expected penalty cost. On the other hand, if the batch jobs do not finish before the required completion deadline, a cost may also be incurred. The allocation is optimal when the cost is minimized. A second example is that, in some areas of the world, electricity cost fluctuates every hour and it is usually cheaper at night [5]. We may wish to save electricity cost (as well as cooling cost which largely consists of electricity cost) during the day by using more servers at night. In such an environment, statically provisioning servers for a batch job cluster would miss the opportunities for savings. It may be desirable to base deployment decisions on the cost function in addition to the demand of workload.

To realize the on-demand provisioning, the overall system must be able to react to changes within a relatively short time. With the use of remote boot images [14], operating system and application installation time can be as short as 5 minutes. In addition, our work [29] reported in Chapter 3 shows that addition or removal of servers can be accomplished in less than one minute. Consequently, server migration from one cluster to another is totally feasible, even if the operating system is rebuilt from scratch. Additionally, for energy-conscious clusters, servers can be remotely turned on/off or woken-up/suspended through the network in seconds. There is no need to keep a cluster with full capacity (all processors running at full speed) when the workload is light. As a result, more effective resource utilization can be achieved by migrating idle servers to other clusters, or simply turning off or suspending the idle servers.

Note that batch jobs have long execution times measured in minutes or hours, so the number of jobs in the system does not change significantly within a short period of time, say on the order of minutes. As a result, decision points, where resource allocation decisions are made, can be scheduled at intervals on the order of 10 minutes. At each decision point, the system must generate a provisioning decision quickly, using the data collected up to that point, so that the system with more or fewer servers will operate for a sufficiently long time before the next decision point is reached.

In this thesis, we are interested in automated resource management for a single batch job cluster hosted in a server farm. Autonomic systems are typically built upon a logical loop of three phases: i) Measure, ii) Decide, and iii) Control. Our focus is on the allocation algorithms used in the Decide phase of the "autonomic loop". Specifically, five new algorithms that determine the number of servers needed in a batch job cluster under different workloads at different times are developed and evaluated. The results will provide the needed information for server provisioning decisions (i.e., add or remove servers). A time-varying cost function for server usage is defined. The merit of our algorithms is measured by the cost incurred to complete all batch jobs by their completion deadline. Our investigation is restricted to the case where all batch jobs have a common, shared deadline.

Our algorithms use a feed forward approach based on a predictive model of a system. This model is used to determine the best allocation of resources. It explores all possible outcomes probabilistically and uses the expected cost as a guide to optimize provisioning decisions. The feed forward approach is different from other studies based on a feedback mechanism [11], which is reactive in nature. Feed forward control has been used to optimize resource allocation in multi-tier web applications [1, 16, 20]. We are interested in testing the effectiveness of this approach for resource provisioning in batch job clusters.

In practice, batch job clusters are often equipped with third-party batch job schedulers that are responsible for dispatching jobs to different nodes of the cluster. For batch jobs with shared deadlines, the impact of the scheduling algorithm is not significant. For simplicity, we use the first-come-first-served (FCFS) scheduling algorithm in our investigation. Contrary to reactive approaches used in other studies [2, 23], we use a predictive approach to estimate the probability of meeting a shared completion deadline for a given number of servers in the cluster. In general, it is difficult to obtain this probability analytically. We therefore use simulation. In our simulation, batch job arrivals are characterized by a job profile and a job's service time requirement is estimated from job execution history. Simulation runs are made to pre-compute the probability of meeting the shared deadline, which can be used to to make server provisioning decisions at decision points.

This thesis makes the following contributions towards building a batch job cluster that manages its own resources.

1. New heuristic algorithms are proposed for deciding the number of servers to use in a cluster at decision points.
2. A time-varying cost function is considered when making resource allocation decisions. This function models the external factors that affect the number of servers to use in a cluster.
3. We have included in our model the delay and cost of increasing or decreasing the number of servers used in the cluster. This is an important factor but omitted in many of the previous work.
4. An implementation of our algorithms based on FCFS scheduling is shown. The different performance aspects of our algorithms are validated by simulation.
5. The proposed resource allocation algorithms are applicable to existing batch job clusters.

1.1 Outline

Chapter 2 discusses related work and organizes it into two categories: provisioning in the full server utility model and provisioning in the shared server utility model. Chapter 3 shows an architecture and some observations of how dynamic server provisioning can be achieved in batch job clusters. Chapter 4 presents the design of our algorithms to determine the number of servers that one should use. In Chapter 5, we present the performance results of our algorithms in terms of the total cost. The number of instances where deployments are made and the issue of scalability are also discussed. Finally in Chapter 6, we conclude our work and give some directions for future research in the area.

Chapter 2

Related work

Previous work in dynamic resource allocation of server clusters can be classified into the full server utility model and the shared server utility model. In the shared server utility model, multiple services are hosted on each server. In contrast, in the full server utility model, each server offers only one service at a time. The work presented in this thesis belongs to the full server utility model.

2.1 Provisioning in the full server utility model

2.1.1 Provisioning in a data centre

In a data centre that employs the full server utility model, a cluster is dedicated to run one application, e.g., webpage hosting, for one customer. Different applications and different customers do not share the same cluster. This model has been implemented in a commercial product [12] which supports autonomic server provisioning in data centres. This product realizes a data centre management method [24] which makes use of virtual LANs and SANs to partition resources into domains called virtual application environments (VAE). Each application is run in its own VAE. Servers are allocated to (or de-allocated from) the VAE according the monitored server utilization.

In [2], a Service-Level-Agreement-based management system is presented. Response time data are gathered for each application cluster. Addition or removal of servers is triggered by the violation of the service level agreement defined in terms of response time. A dynamic resource allocation algorithm to maintain a target cluster-wide average CPU utilization in a data centre is presented in [23]. This target is attained by acquiring and releasing servers in response to changes in load. The algorithm presented assumes a linear relationship between response time and CPU utilization.

A method to provision databases used in dynamic content web servers is shown in [27]. In the 3-tier architecture, an autonomic manager tier is interposed between the application server(s) and the database cluster. The autonomic manager tier virtualizes the database cluster, so that the application server sees a single database. Per workload query latency is used as a metric to trigger database allocations. When the latency exceeds the value specified in the Service Level Agreement, extra replicas of the database serving the workload are created.

All of the above provisioning methods are concerned with managing interactive job clusters. For such clusters, the provisioning goal is to provide a target response time percentile with as few resources as possible; response time and CPU utilization are popular metrics.

2.1.2 Provisioning in a grid environment

Resource provisioning in a grid environment for workflows that consist of batch jobs with execution dependencies has been presented in [26]. The objective of that work is to minimize the completion time of workflows. Resource provisioning is done by advance reservation at different grid sites in order to minimize the waiting time of batch jobs in the grid.

Another example of grid provisioning is Cluster-on-Demand (COD), a cluster operating system framework for mixed-use clusters [8, 21]. COD introduces the concept of a virtual cluster, which is a functionally isolated group of servers. A key element of COD is a protocol to resize virtual clusters dynamically by making use of the Sun GridEngine (SGE) [28], a batch job scheduler for grids. Physical servers can be added to (or removed from) the virtual cluster by linking to (or delinking from) the SGE. Provisioning decisions are made by a Virtual Cluster Manager (VCM) based on some pre-specified metrics or policies.

2.2 Provisioning in the shared server utility model

In the shared server utility model, servers are shared among different customer priority classes. Generally, higher priority class customers require a lower response time percentile. Each server may run more than one kind of services, e.g., webpage hosting and database hosting. The services hosted are interactive jobs which share the CPU simultaneously. Dynamic provisioning systems of this type can be categorized by their provisioning goals: maintaining the quality of service, maximizing the profit from customers, and minimizing the electricity cost.

Maintaining the quality of service. The general objective to maintain the quality of service (QoS) is to deliver better services to higher priority classes of customers without over-sacrificing low priority classes. A certain amount resources is reserved to the lower priority classes to avoid over-sacrificing. In [3, 30], algorithms for providing differential service to customers of different priority classes are presented. These algorithms control, for each server, the amount of CPU time allocated to each customer. CPU allocation in

[3] is achieved by techniques discussed in [4], which work at kernel level of the operating system. In contrast, the work reported in [30] is at the level of the operating system API. In [32], the QoS for different customer requests is maintained by a scheduling algorithm at the network switch. Servers in the cluster are dynamically partitioned to serve different requests according to the workload and priority.

Maximizing the profit from customers. In our discussion, profit refers to the amount of economic benefit gained by the data centre by serving requests in a timely fashion. It is often defined in the service level agreement (SLA) in terms of response time performance. SLAs of higher priority classes guarantee a shorter response time, but incur a higher cost to the customers. For the data centre, revenue is gained by satisfying the SLAs, and a penalty is paid otherwise. In [25, 31] a decentralized approach that schedules requests of different priority inside the request queues of each server is discussed. In comparison, algorithms presented in [15, 17] make use of the network switch or gateway to partition the cluster resources.

Minimizing the electricity cost. Energy-conscious systems are concerned with saving electricity by using the smallest possible amount of CPU power. In [6], it was mentioned that the CPU is the largest consuming component for typical web servers. Energy-saving can be achieved by directing requests to a minimal active set of servers at the network switch and keeping idle servers in low-power states [7]. Authors of [9] further refines the techniques by enabling CPUs to operate at different frequencies, which are linearly proportional to energy consumption and inversely proportion to the response time of the system.

As a summary, resource sharing in the shared server utility model is mainly implemented by two mechanisms: server multiplexing and switch redirection. Server multiplexing refers to multiplexing server resources (CPU time) among the hosted applications. Kernel modification is usually required. Switch redirection refers to redirecting different requests to different logical groups of servers inside a cluster by a network switch.

2.3 A general resource provisioning framework

The IBM Tivoli Intelligent Orchestrator (TIO) [13] is an autonomic engine that orchestrates resources of a data centre among different application clusters. As described in [19], it consists of four core components: data acquisition engine, objective analyzer, resource broker, and deployment engine. The decision cycle of TIO is illustrated in Figure 2.1. The data acquisition engine collects performance data (e.g., CPU utilization) from the monitored application cluster. The data is input to the objective analyzer which compares the data to the service level objective of that application. It also calculates the probability of breaching (PoB) the service level objective and this probability is reported to the resource broker. The resource broker makes resource allocation decisions according to the received PoB values of different application clusters. Finally, server deployment or removal is carried out by the deployment engine. In TIO, users can implement their own objective analyzer that estimates PoB using different metrics, e.g., transactions per second in the case of database clusters.

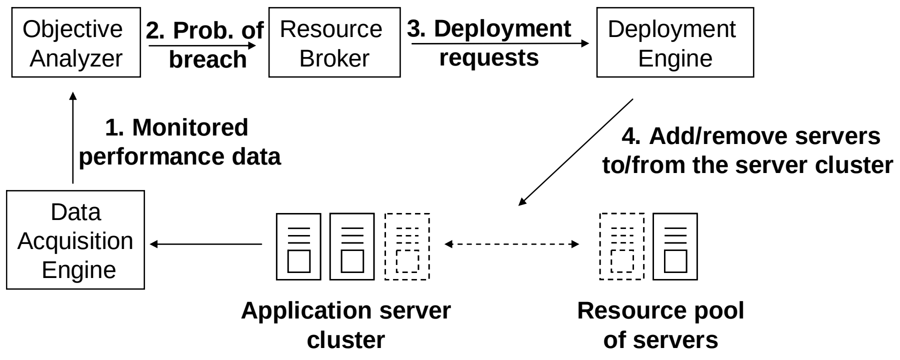

Figure 2.1: Decision cycle of TIO

Chapter 3

Performance measurement of server deployment and removal

3.1 Overall approach

Autonomic systems usually implement a logical loop consisting of three phases: i) Measure, ii) Decide, and iii) Control (see Figure 3.1). The loop is performed periodically. In this thesis, we are interested in the Decide phase for a batch job cluster where server nodes can be dynamically added or removed. In the Measure phase, state information such as the number of jobs and the number of servers in the system is measured. For the Decide phase, heuristic algorithms for dynamic resource provisioning are developed. These algorithms compute the information required for provisioning decisions, using the data obtained in the Measure phase as input. The algorithms then decide whether to change the number of servers deployed in the cluster or not. In the Control phase, the system implements the change, if any.

An important requirement for the Control phase to work properly is that the delay to

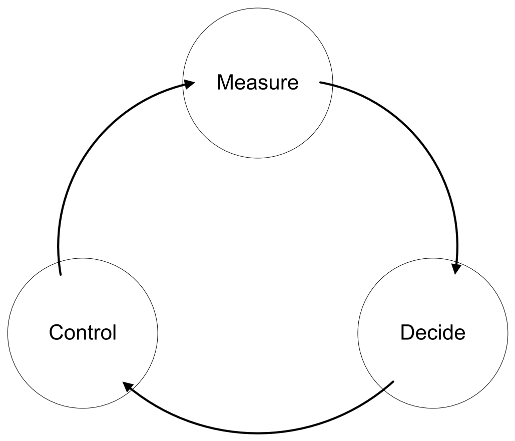

Figure 3.1: The 3 phases of the “autonomic loop”

add a server to or remove a server from the cluster is not excessive. In this chapter, we implement a proof-of-concept prototype that supports autonomic resource provisioning and use this prototype to obtain measurement data regarding the delay in server deployment or removal.

3.2 Proof-of concept prototype

The architecture of our prototype is based on TIO, as illustrated in Figure 3.2. Our prototype is capable of provisioning heterogeneous server nodes in a cluster. It extends the four core components of TIO described in Section 2.3. A fifth component, the job history database, is added to facilitate implementation. The details of these five components are described below.

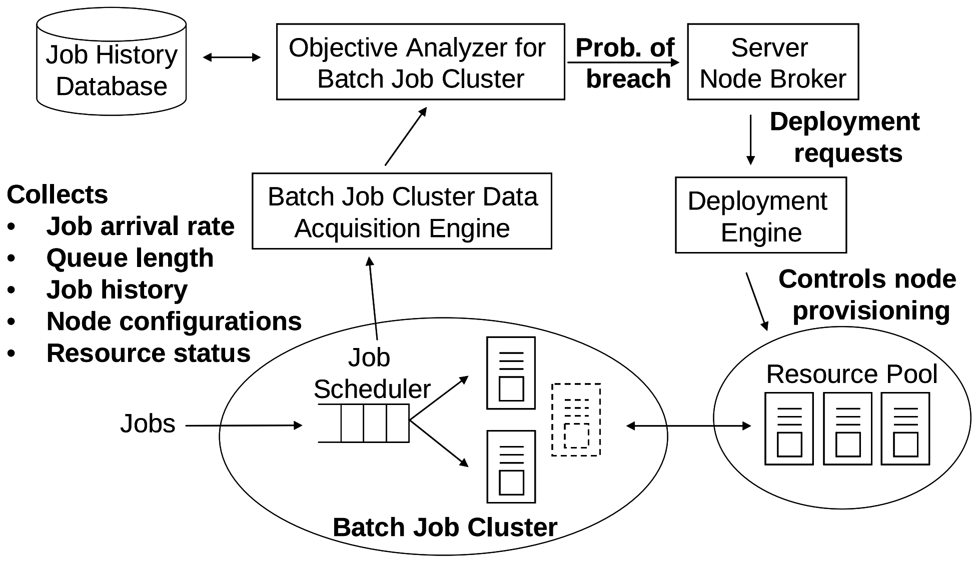

Figure 3.2: Prototype architecture

3.2.1 Data acquisition engine

The data acquisition engine collects data from the cluster. These include job arrival rate, queue length, job history, configuration of server node and their resource status (e.g., processor utilization, memory usage, disk usage and response time). The data acquisition engine periodically reports the resource status to the objective analyzer and the objective analyzer stores the data in the job history database.

3.2.2 Job history database

The job history database stores data such as the job submission time, waiting time, processing time, completion time, execution server and other job details. These data will be used as input when there is a need to predict workload parameters such as arrival rate and service time.

3.2.3 Objective analyzer

The objective analyzer periodically computes the probability of breaching the service level requirement as a function of the number of servers deployed. This probability of breach (PoB) is reported to the resource broker. At the objective analyzer, the data in the job history database are used to estimate the future workload. This estimated workload and the current performance data reported by the data acquisition engine are used as input to compute the PoB. For our prototype, the service level requirement is related to completion of batch jobs before deadline.

3.2.4 Resource broker

The resource broker receives the PoB periodically from the objective analyzer. When the PoB reported is higher than the threshold defined in reference to the service level agreement, one or more servers may be deployed. On the other hand, if the PoB is below the threshold, one or more servers may be removed. The resource broker determines when the cluster should be allocated more or fewer server nodes.

3.2.5 Deployment engine

The deployment engine carries out the deployment (or removal) of servers as per decisions by the resource broker. For ease of implementation, we use an existing job scheduler called the Condor scheduler. One way to deploy (or remove) servers from the cluster is to link (or delink) the server to the job scheduler. Once the node is linked, the scheduler will dispatch jobs to the newly linked node according to its scheduling algorithm. When the node is delinked, the scheduler will stop dispatching jobs to the delinked node.

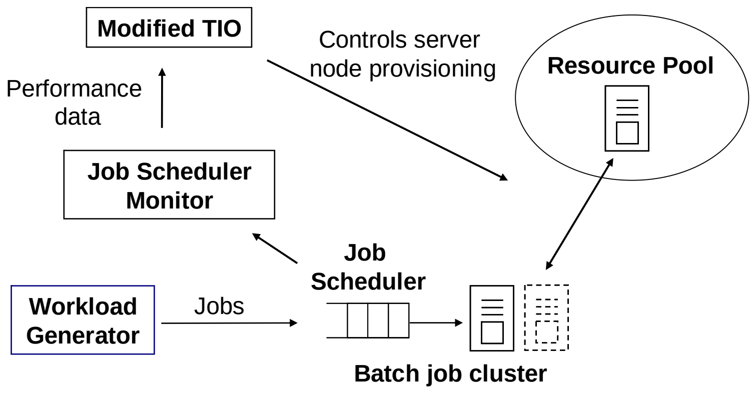

Figure 3.3: Experiment environment

3.3 Performance measurement

We have implemented a prototype based on the architecture described above. Our prototype is shown in Figure 3.3. This cluster consists of one server initially. The Condor scheduler and the data acquisition engine are hosted at this node. There is one other server in the resource pool. This server can be allocated if required.

The data acquisition engine reports the following data every five seconds to the TIO server, which runs on a separate machine: number of jobs in queue, job arrival rate, and service time of completed jobs. Information collected through the Condor scheduler includes the current queue length, and submission and completion time of all arrived jobs. Arrival rate and service time are then calculated based on the reported information from the queries.

An objective analyzer designed for batched jobs with a common completion deadline was implemented and installed in the TIO server. The resource broker provided by TIO was modified such that it would understand the probability of breach reported by the objective analyzer. The deployment engine deploys or removes server nodes by linking or delinking the nodes to the Condor scheduler.

In our experiment, a workload profile is used to specify the arrival rate of batch jobs as a function of time. Based on this profile, a load generator creates batch jobs and submits them to the scheduler. The service requirement of each arriving job is estimated using data in the job history database. The batch jobs are executed at the cluster according to the algorithm used by the Condor scheduler.

We used a simple approach to calculate PoB in the objective analyzer. PoB is defined to be the predicted fraction of jobs missing deadline. The predicted values are obtained by simulating a batch cluster.

By experimenting with our prototype, we found that the Condor scheduler took about 25 seconds to add a server node, but required about 30 seconds to remove a node. Overhead is incurred when a server node is added or removed because the server is not able to execute jobs during this time. Since batch jobs usually have a long run time (much larger than 30 seconds) and the common deadline could be as late as the next morning, an overhead of 25 to 30 seconds is not significant. In our resource pool, we assume that nodes already have all the required software pre-installed to process jobs. If the nodes to be added do not meet the software pre-requisite, additional software should be installed by the deployment engine before linking the nodes to the scheduler. In practice, different clusters may require a different set of software or operating system for their nodes. In that case, software or operating system images are usually prepared for nodes that are shared among the clusters. Upon server deployment, the images can be loaded to the nodes through the network in an efficient manner.

Moreover, the Condor scheduler enables checkpointing of jobs. The status of jobs is automatically monitored and stored in a checkpoint server. Since a server may be migrated out of the cluster when it is processing a job, checkpointing allows the job to be migrated to another server without wasting the CPU time invested. For batch job clusters where checkpointing is not available, a server may need to finish the current job before it can be removed, incurring a higher delay in removal.

3.4 Remarks

The objective of this thesis is to design and evaluate algorithms for autonomic resource provisioning in a batch job cluster. The prototype provided valuable information on the delay required to add or remove a processor node. This information will be useful in algorithm design. However, the prototype is implemented on a cluster of two server nodes. This is not suitable for scenarios where it is desirable to deploy more than two servers. For this reason, we will use simulation to investigate the effectiveness of our resource provisioning algorithms.

Chapter 4

Heuristic algorithms for dynamic resource provisioning

4.1 Performance model

In this section, we present our algorithms for autonomic resource provisioning in a batch job cluster.

We first develop a performance model that will be used in our investigation. In this model, the batch job cluster is assumed to be dynamically configurable with at least $p_{min} \geq 1$ and at most p_{max} servers. These servers are identical and have the same capacity with respect to processing batch jobs. Each server is assumed to execute only one batch job at a time. This assumption is consistent with the default behaviour of popular batch job schedulers, e.g., Condor [10], PBS [22] and LSF [18]. It follows that batch jobs do not share processors and jobs are queued at the scheduler until there is a free processor.

The operation of the batch cluster is organized in job processing periods. The activities within a period are illustrated in Figure 4.1. At the beginning of the period (or time 0), the cluster consists of p_{min} clusters. The time unit corresponds to time interval between decision points. The decision points therefore occur at time = 1, 2, ... Submission of batch jobs happens in the job submission period only. This period starts at time 0 and ends at time u, the submission deadline. All batch jobs are assumed to have a common deadline which occurs at time d. This is referred to as the shared deadline. Our objective is to keep the probability that all jobs are finished by time d not smaller than that specified by the service level agreement. The issue of how jobs not meeting the deadline are handled is beyond the scope of this thesis.

Maintaining the service level agreement regarding the completion deadline has a higher priority than reducing cost. We thus consider a service level agreement that specifies all jobs are completed on time. Once there is a lack of confidence that all jobs will be finished on time, servers should be added to the cluster. Note that the cluster cannot have more than p_{max} servers. Therefore, no further server can be added when p_{max} is reached. Since there is no job arrival after the submission deadline, we expect that some servers can be removed when the number of jobs in the system is less than the number of servers. In our model, even when there is no job in the system, the number of servers in the cluster is at least p_{min}.

We assume that the arrival rate of jobs is time-dependent during the job submission period and that the service time of jobs is independent and exponentially distributed.

Three time periods are defined when a server is deployed: "deployment period → usage period → removal period" (see Figure 4.2). The deployment period (denoted by r_1) starts when a decision is made to add one or more servers. It models the time required to load the necessary software and configurations in order to add one or more servers to the cluster. The length of the deployment period is not affected by the number of servers to be added at the same time. Jobs are not processed by the additional servers during this period. The

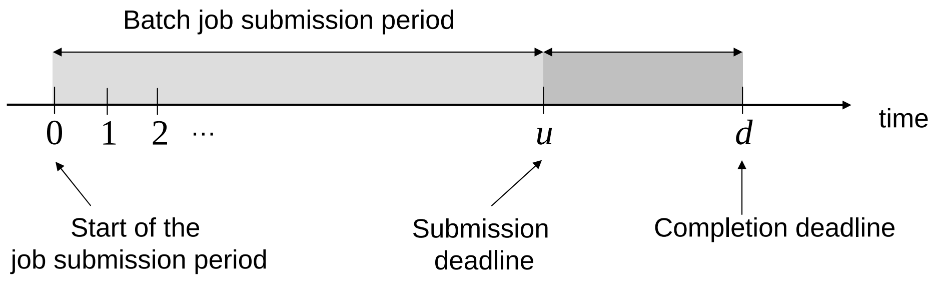
Batch job submission period
time
0
1
2
···
u
d
Start of the job submission period
Submission deadline
Completion deadline

Figure 4.1: Job processing period

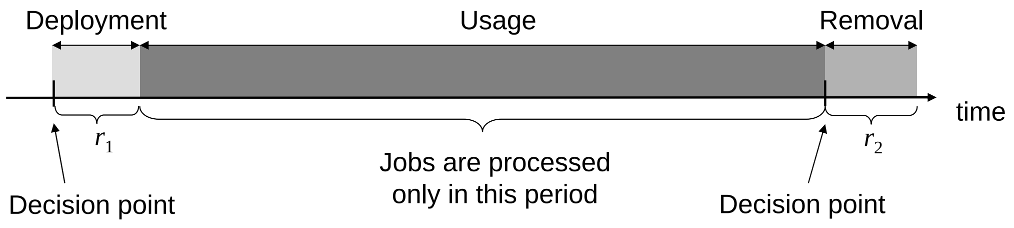

Figure 4.2: Server deployment period

usage period models the time period during which the newly added servers are used to process jobs. The removal period (denoted by r_2) starts when a decision is made to remove one or more servers. It models the time required to remove the servers from the cluster. We assume that job checkpointing is enabled, so waiting for the current job to finish before server removal is not required.

Servers are added or removed during a job processing period. An example of the number of servers in the cluster throughout a job processing period is shown in Figure 4.3. Once the deployment period starts, the additional servers are considered part of the cluster until the end of the removal period.

Each server has an identical cost function c_t that gives the instantaneous cost at time instant t for keeping the server in the cluster. The cost function may be time-varying; an

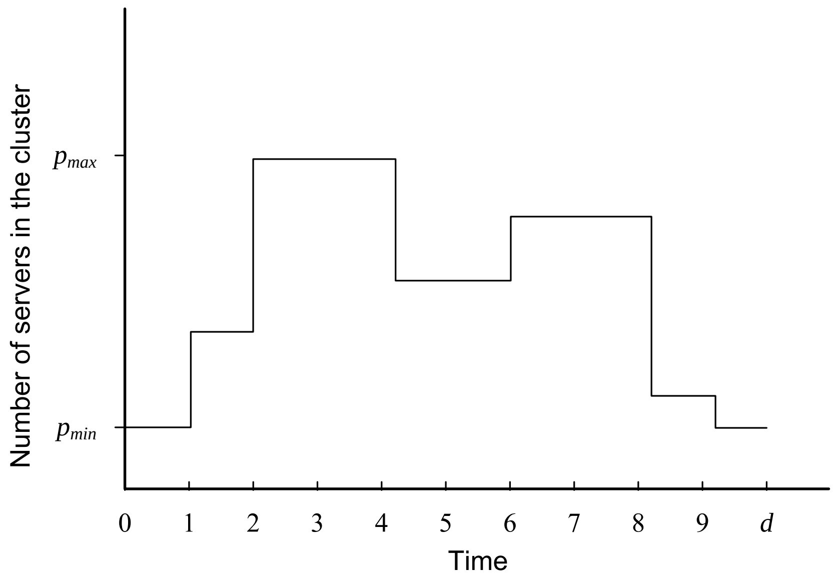

Figure 4.3: An example of the number of servers in a cluster throughout a job processing period

example is depicted in Figure 4.4. A server is considered to be in use from the start of the deployment period to the end of the removal period (see Figure 4.2). Let p_t be the number of servers used in the cluster at time t. The total cost is given by:

$$C = \int_0^d [c_t \times p_t] \; dt \tag{4.1}$$

The merit of our resource provisioning algorithms is evaluated with respect to C.

4.2 Simple heuristic algorithms

We first propose two heuristic algorithm resource provisioning algorithms that do not take the cost function into account, but aim at finishing all jobs by the deadline. By comparing the performance of these algorithms with those that take into consideration

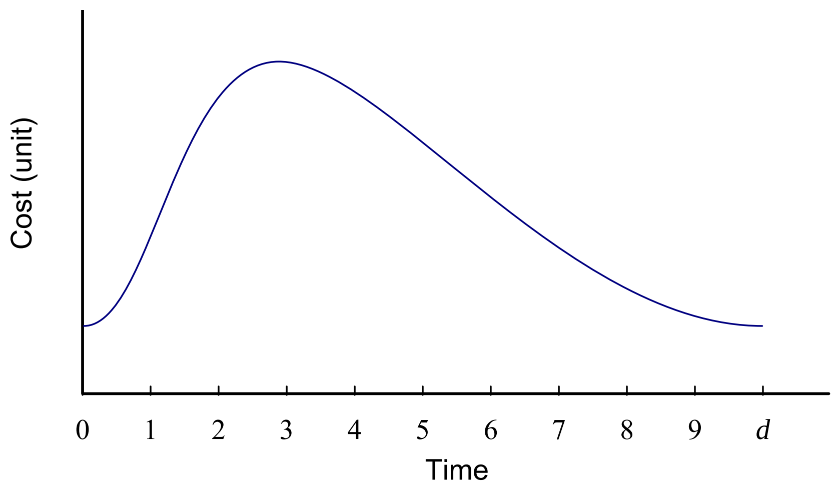

Figure 4.4: An example of a cost function throughout a job processing period

the cost function (to be defined in Section 4.3), the benefits of using cost-function-based algorithms can be assessed.

4.2.1 Heuristic Algorithm 1: Threshold-responding deployment heuristic algorithm

This algorithm makes deployment decisions based on an estimated percentage of jobs that are completed on time. Recall that p_t is the number of servers used at time t. Let n_t be the number of jobs at time t. Consider a decision point at time s. Measured data for n_s and p_s are obtained at the Measure phase. At the Decide phase, we determine whether we are confident that all jobs will be finished on time if no change is made to the number server used. If there is insufficient confidence, extra servers are needed. On the other hand, servers are removed if we can achieve the required level of confidence by using fewer servers.

Since our performance model is stochastic in nature, it may not be possible to achieve a 100% probability that all jobs are completed on time (e.g., with a very small probability, the service time of a job may be sufficiently long that the deadline is missed even though execution of this job is started immediately). We therefore use the following condition to reflect the service level agreement:

$$\Pr[\text{all jobs are finished on time}] > y\% \tag{4.2}$$

where y is very close to 100 (in this thesis, we use $y = 99.99$).

Suppose the same number of servers, say p, is used from time s to time d. A larger n_s would mean a smaller probability that all existing and future jobs are completed before the shared deadline d. Let $g_s(p)$ be the largest value of n_s such that this probability is larger than or equal to $y\%$. $g_s(p)$ is obtained as the solution to the following relations:

$$\Pr[n_d = 0 | n_s = g_s(p) \text{ and } p_i = p \text{ for } s \leq i \leq d] \geq y\% \tag{4.3}$$

and

$$\Pr[n_d = 0 | n_s = g_s(p) + 1 \text{ and } p_i = p \text{ for } s \leq i \leq d] < y\% \tag{4.4}$$

In general, using more servers means having more processing capacity and should lead to a higher probability that all jobs are completed on time. We thus expect $g_s(p)$ to be a non-decreasing function of p. At decision point s, let v_s be the minimum number of servers required to achieve a $y\%$ confidence in finishing all jobs by time d.

$$v_s = p \text{ such that } n_s \leq g_s(p) \text{ and } n_s > g_s(p-1) \tag{4.5}$$

At the Decide phase, the decision to change the number of servers from p_s to v_s. The algorithm that implements this decision (Heuristic Algorithm 1) is shown below.

Algorithm 1 Heuristic Algorithm 1: Threshold-responding deployment heuristic algorithm

```
Ensure: The number of servers to add or remove. Positive return value means add,
    negative means remove, 0 means no change.
1: Determine $v_s$ from Equation 4.5.
2: if $v_s > p_{max}$ then
3:     $v_s = p_{max}$
4: else if $v_s < p_{min}$ then
5:     $v_s = p_{min}$
6: end if
7: return $v_s - p_s$
```

4.2.2 Heuristic Algorithm 2: Delayed threshold-responding deployment heuristic algorithm

Heuristic Algorithm 2 is a variant of Heuristic Algorithm 1. The motivation is to minimize the fluctuation in the number of servers used. This is accomplished by delaying the removal decision. Specifically, a decision to remove servers is carried out only if the condition for removal (i.e., $v_s - p_s < 0$) is met for two consecutive decision points. This algorithm is shown in Algorithm 2.

4.2.3 Method to determine v_s

We note from the descriptions of Heuristic Algorithms 1 and 2 that we need to determine v_s. v_s, as defined in Equation 4.5, is a function of $g_s(p)$ and n_s. We must therefore first determine $g_s(p)$. This is accomplished by simulating a single queue, multiple server model with FCFS discipline. The inputs to the simulation are:

- n: number of jobs at time s
- p: number of servers in the cluster when time s is reached

Algorithm 2 Heuristic Algorithm 2: Delayed threshold-responding deployment heuristic algorithm

```
Ensure: The number of servers to add or remove. Positive return value means add,
    negative means remove, 0 means no change.
1: Determine $v_s$ from Equation 4.5.
2: if $v_s > p_{max}$ then
3:     $v_s = p_{max}$
4: else if $v_s < p_{min}$ then
5:     $v_s = p_{min}$
6: end if
7: if $v_s \geq p_s$ or $(v_{s-1} < p_{s-1}$ and $v_s < p_s)$ then
8:     return $v_s - p_s$
9: else
10:    return 0
11: end if
```

- Arrival rate of batch jobs is based on a job profile, which may be time-varying. An example is given in Figure 4.5, where time 7 represents the submission deadline.
- Service time is exponentially distributed

We run the simulation for a large number of runs from time s to d for different combinations of n and p. We assume that each server starts processing a new job at time s. We collect data for the fraction of runs that all jobs are completed by the deadline. This fraction (denoted by $Q_s(p, n)$) is used as an estimate for the probability that all jobs are completed by the deadline. $g_s(p)$ then is given by the value of n such that $Q_s(p, n) \geq y\%$ and $Q_s(p, n + 1) < y\%$ (see Equations 4.3 and 4.4). We use a binary search to obtain the value of $g_s(p)$ at time s. The initial minimum number of jobs to try is 0 and the maximum number is:

$$\gamma = p_{max} \times \frac{d}{\text{mean service time}} \tag{4.6}$$

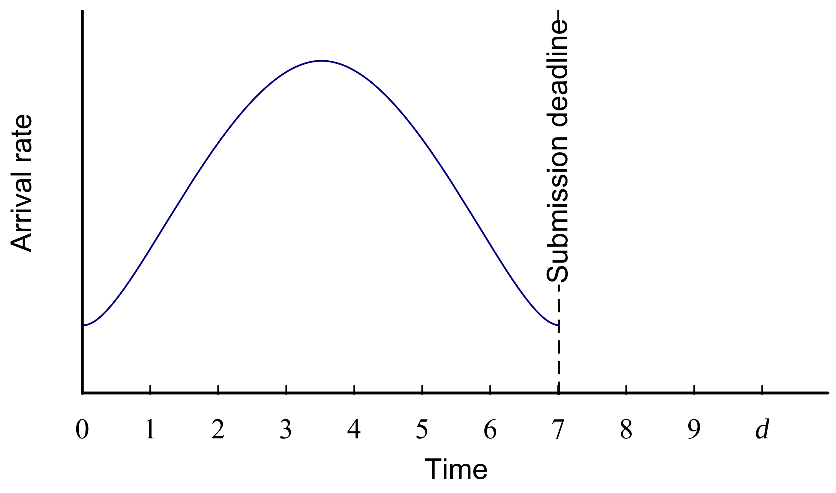

Figure 4.5: An example of an arrival profile

For a given p, the complexity to find values of $g_s(p)$ for different s is $O(d \ln \gamma)$, where d is the total number of decision points and γ is given by Equation 4.6.

The above procedure allows us to pre-compute $g_s(p)$ for different combinations of p and s. At the Decide phase, $g_s(p)$ can simply be obtained by a table-lookup.

4.3 Cost-aware resource allocation algorithms

In this section, we present two additional heuristic algorithms which aim at minimizing the total cost (as defined in Equation 4.1) and finishing all jobs on time. This requirement is again written as Pr[all jobs are finished by the deadline] $> y\%$ ($y = 99.99$). At decision point s, we need to determine l_s, the number of servers that would yield the lowest expected cost if this number of servers is used from time s to $s + 1$. The decision is then to change the number of servers from p_s to l_s.

4.3.1 Preliminary remarks

The following results are useful in our analysis of estimated cost:

- We note from the discussions in Section 4.2.1 that we can determine, for decision point s, the value of $g_s(p)$, which is the largest value of n_s such that Pr[all jobs are finished by the deadline] $> y\%$ ($y = 99.99$) given that p servers are used from time s to d.

- Once the submission deadline (at time u) has been reached, there are no more job arrivals. Some of the servers will be idle if the number of servers is larger than the number of jobs in the cluster. So, at decision point s, $u \leq s < d$, the maximum number of servers needed (denoted as w_s) is not more than n_s.[1] Since we must have at least p_{min} and cannot have more than p_{max} servers in the cluster, we can write,

$$w_s = \min\{p_{max}, \max\{p_{min}, n\}\} \text{ for } u \leq s < d. \tag{4.7}$$

4.3.2 Analysis of estimated cost

We define an estimated cost which will provide the needed information to determine l_s for $s < d - 1$, as follows:

$$\begin{aligned} L_s(p, n) = {} & \text{estimated cost from } s \text{ to } d \text{ given that the values of} \\ & n_s \text{ and } p_s \text{ at decision point } s \text{ are } n \text{ and } p \text{ respectively.} \end{aligned} \tag{4.8}$$

In this subsection, we discuss how to find $L_s(p, n)$. Note that p_s refers to the number of servers when time s is reached. Analytic results for l_s, where $s < d - 1$, will be presented

[1]Note that we may use fewer servers than the maximum number of servers needed.

in Sections 4.3.3 when we describe our heuristic algorithms.

$L_s(p, n)$ can be obtained recursively backwards; the base case being $s = d - 1$.

Base case

For $s = d - 1$, $L_{d-1}(p, n)$ has 3 components.

1. The first component is the cost of using l_{d-1} servers from time $d - 1$ to d. This is given by $l_{d-1} \times \int_{d-1}^{d} c_t \, dt$. For the special case of $s = d-1$, l_{d-1} can be determined by the n_{d-1}. Since there are no more future decision points, we must use enough servers to ensure that Pr[completing the n_{d-1} jobs on time] $> y\%$. Note that one server can finish more than one jobs from time $d-1$ to d, a smaller number of servers than w_{d-1} may be sufficient. We thus have:

$$l_{d-1} = \min\{w_{d-1}, \alpha\} \tag{4.9}$$

   where α is the solution to the following relations:

$$n_{d-1} \leq g_{d-1}(\alpha) \text{ and } n_{d-1} > g_{d-1}(\alpha - 1)$$

2. The second component refers to the cost associated with removing servers. Each removed server incurs a cost of $\int_{d-1}^{d-1+r_2} c_t \, dt$ because a server to be removed is not released until the end of the removal period which has length r_2.

3. The third component is the expected penalty of missing the job deadline. Suppose, for any job, the cost of missing the deadline is P. We assume that P is given by:

$$P = x \times c_{max} \tag{4.10}$$

where x is the mean service time, and $c_{max} = \max_{t=0,\dots,d} c_t$.

A maximum value of c_t is used because the objective is to finish the job by the deadline, and the use of c_{max} in Equation 4.10 will ensure that this objective is met. If $P < x \times c_{max}$, leaving a job unfinished may incur a lower expected cost than processing it, which is not practical.

Recall that n_d is the number of jobs at deadline d (these jobs have missed the deadline). The value of n_d, say m, is affected by n, the value of n_s at $s = d - 1$. Let $k_s(q, n, m)$ be the probability that $n_{s+1} = m$ given that $n_s = n$ and q servers are used from s to $s + 1$. Note that q may be different from p because servers may be added or removed at decision point s. For the base case, l_{d-1} servers are used from time $d - 1$ to d. Pr[m jobs missing deadline (or $n_d = m$)] $= k_{d-1}(l_{d-1}, n, m)$. Since each job incurs a penalty P, the total penalty of m jobs missing the deadline is mP. Summing over all possible values of m, the expected penalty is:

$$\sum_{m=0}^{\infty} k_{d-1}(l_{d-1}, n, m) mP \tag{4.11}$$

Combining the three components discussed above, we have for the base case:

$$L_{d-1}(p, n) = l_{d-1} \times \int_{d-1}^{d} c_t \, dt + L^* + \sum_{m=0}^{\infty} k_{d-1}(l_{d-1}, n, m) mP \tag{4.12}$$

where

$$L^* = \begin{cases} 0 & \text{if } l_{d-1} \geq p \text{ (i.e., servers are added or no change),} \\ (p - l_{d-1}) \times \int_{d-1}^{d-1+r_2} c_t \, dt & \text{if } l_{d-1} < p \text{ (i.e., servers are removed).} \end{cases}$$

Recursive relationship between L_s and L_{s+1}

We distinguish between two cases: $s < u$ (before submission deadline) and $s \geq u$ (at or after submission deadline). Consider case 1. $L_s(p, n)$ can be obtained as follows. Suppose at decision point s, the number of servers is changed from p to some other value q (no change if $p = q$). To characterize q, we note that if $n \geq g_s(p_{max})$, we do not have sufficient confidence that all jobs will be completed on time. We therefore use the maximum number of processors available; it follows that $q = p_{max}$.

On the other hand, if $n < g_s(p_{max})$, the value of q that yields the minimum $L_s(p, n)$ will be selected. Define an auxiliary function $M_s(p, q, n)$ which is the estimated cost from s to d given n jobs at time s and the number of servers used is changed from p to q at time s. $M_s(p, q, n)$ is the sum of:

1. the cost of using q servers from time s to $s + 1$,
2. the estimated cost from time $s + 1$ to d, and
3. the cost associated with adding or removing servers at time s, depending on whether $q > p$ or $q < p$.

We thus have

$$M_s(p, q, n) = q \times \int_s^{s+1} c_t \; dt + \sum_{m=0}^{\infty} k_s(q, n, m) \times L_{s+1}(q, m) + M^* \tag{4.13}$$

where

$$M^* = \begin{cases} 0 & \text{if } q > p, \\ (p - q) \times \int_s^{s+r_2} c_t \; dt & \text{if } q \leq p. \end{cases}$$

The relationship between L_s and L_{s+1} can now be written as:

$$L_s(p,n) = \begin{cases} \min_{q=p_{min},\ldots,p_{max}} M_s(p,q,n) & \text{if } n < g_s(p_{max}), \\ M_s(p,p_{max},n) & \text{if } n \geq g_s(p_{max}). \end{cases} \tag{4.14}$$

We next consider case 2, i.e., $s \geq u$. As discussed in Section 4.3.1, the maximum number of servers needed after the submission deadline (time u) is w_s. Therefore, we use w_s instead of p_{max} servers if $n_s \geq g_s(p_{max})$. Hence, for $s \geq u$, $L_s(p,n)$ can be written as:

$$L_s(p,n) = \begin{cases} \min_{q=p_{min},\ldots,w_s} M_s(p,q,n) & \text{if } n < g_s(p_{max}), \\ M_s(p,w_s,n) & \text{if } n \geq g_s(p_{max}). \end{cases} \tag{4.15}$$

This completes our analysis of the estimated cost $L_s(p,n)$.

4.3.3 Heuristic Algorithm 3: Cost-aware deployment heuristic algorithm

The key step of our cost-aware deployment heuristic algorithm is to determine l_s for $s = 1,\ldots,d-1$. When $s = d-1$, analytic results for l_{d-1} have been presented in Section 4.3.1. Specifically, l_{d-1} is given by Equation 4.9.

Consider next the case $s < d-1$. For a given $L_s(p,n)$, l_s can be obtained from Equations 4.14 and 4.15. There are two subcases: $s < u$ and $s \geq u$. For $s < u$,

$$l_s = \begin{cases} q^* & \text{if } n < g_s(p_{max}), \\ p_{max} & \text{if } n \geq g_s(p_{max}) \end{cases} \tag{4.16}$$

where $M(p,q^*,n) = \min_{q=p_{min},\ldots,p_{max}} M(p,q,n)$.

For $s \geq u$,

$$l_s = \begin{cases} q^* & \text{if } n < g_s(p_{max}), \\ w_s & \text{if } n \geq g_s(p_{max}) \end{cases} \tag{4.17}$$

where $M(p, q^*, n) = \min_{q=p_{min},\ldots,w_s} M(p, q, n)$.

Once l_s has been determined, decision to add or remove servers is made as follows.

1. Add $l_s - p_s$ servers if $l_s > p_s$
2. Remove $p_s - l_s$ servers if $l_s < p_s$
3. Otherwise: no change

This is shown in Algorithm 3.

Algorithm 3 Heuristic Algorithm 3: Cost-aware deployment heuristic algorithm

Ensure: The number of servers to add or remove. Positive return value means add, negative means remove, 0 means no change.

```
1: At decision point $s$, use the measured values of $p_s$ and $n_s$ (denoted by $p$ and $n$) as
   input to determine $l_s$ from Equations 4.9, 4.16 and 4.17.
2: return $l_s - p$
```

4.3.4 Method to compute $L_s(p, n)$

In this section, we discuss how we may compute $L_s(p, n)$. Based on the results in Section 4.3.2, $L_s(p, n)$ can be computed backwards from $s = d - 1$. This computation is shown in Algorithm 4. We note from Equations 4.12, 4.14 and 4.15 that $L_s(p, n)$ and the auxiliary function $M_s(p, q, n)$ are expressed in terms of $k_s(q, n, m)$. We must therefore first determine $k_s(q, n, m)$. Recall that $k_s(q, n, m)$ is the probability that $n_{s+1} = m$ given that $n_s = n$ and

Algorithm 4 Algorithm for computing $L_s(p,n)$

```
1: Initialize $L_{d-1}(p,n)$ for $p_{min} \leq p \leq p_{max}$, $0 \leq n \leq g_{d-1}(p_{max})$.
2: for $s = d-2$ to 1 do
3:    for $n = 0$ to $g_s(p_{max})$ do
4:       for $p = p_{min}$ to $p_{max}$ do
5:          Compute $L_s(p,n)$ according to Equations 4.12, 4.14 and 4.15.
6:       end for
7:    end for
8: end for
```

q servers are used from s to $s+1$. $k_s(q,n,m)$ can be determined by using the simulation described in Section 4.2.3. To reduce the amount of computation, we estimate the values $k_s(q,n,m)$ as follows.

- We run the simulation from 1,000 time 0 to d for each different value of q. The number of jobs at time 0 is set to a large number such that the number of jobs in the system does not fall below q throughout the simulation. Our experience shows that a sufficiently large value is given by: $p_{max} \times \frac{d}{\text{mean service time}} \times 2$. From the simulation, we determine D_s as the largest recorded decrease in number of jobs from s to $s+1$ ($D_s = 0$ if no decrease is recorded) for $s = 0, 1, \ldots, d-1$.

  Let $R_s = q + D_s$. For $n_s \geq R_s$, we assume that the number of jobs from time s to $s+1$ is at least q. When this happens, the change in number of jobs from time s to $s+1$ is independent of n_s. Let $\tilde{\delta}_s = n_{s+1} - n_s$. During the simulation, we collect data for the probability distribution of $\tilde{\delta}_s$ for $s = 0, 1, \ldots, d-1$. For $n \geq R_s$, we estimate $k_s(q, n, n+\delta)$ by $\Pr[\tilde{\delta}_s = \delta]$.

- For $n < R_s$, we run a special simulation from time s to time $s+1$ with all possible combinations of decision point s, number of servers q and number of jobs n. For each combination, data is collected for the probability distribution of $k_s(q,n,m)$.

The above procedure allows us to pre-compute $k_s(q, n, m)$ for all combinations of s, q, n and m. These results are then used to compute $L_s(p, n)$.

$L_s(p, n)$ is also pre-computed. Once values for $k_s(q, n, m)$ are available, the steps outlined in Algorithm 4 can be used to compute $L_s(p, n)$.

In our computation of $L_s(p, n)$, the following method is used to improve efficiency. At the beginning of the job processing period, the number of jobs in the system is expected to be small. Suppose a number K_s can be found such that $\Pr[n_s > K_s] \approx 0$. Then, instead of pre-computing all values of $L_s(p, n)$ or $L'_s(p, n)$ from $n = 0$ to $g_s(p_{max})$, we can pre-compute only the values from $n = 0$ to $\min\{g_s(p_{max}), K_s\}$, which would save time. K_s can be set to be the summation of the maximum increase in jobs between every two consecutive decision points from time 0 to s observed in the simulation to obtain $k_s(q, n, m)$.

To compute $L_s(p, n)$, the time complexity is $O(dJG)$, where d is the total number of decision points from time 0 to the deadline, $J = p_{max} - p_{min}$ and $G = \max_{s=0}^{d-1} g_s(p_{max})$. Note that the amount of computation can be excessive for large values of J and G. In this regard, we introduce two modified version of Heuristic Algorithm 3, designed to reduce the amount of computation required. They are referred to as Heuristic Algorithms 4 and 5; it will be described in the next two sections.

4.3.5 Heuristic Algorithm 4: Cost-aware deployment heuristic algorithm with modified L_s

Consider the computation of $L_s(p, n)$. The modified version of Heuristic Algorithm 3 is based on the following observation. Let β be the number of servers used between s and $s+1$ given that $n_s = n-1$. Intuitively, when $n_s = n$, the number of servers needed (or l_s) should be no less than β. Therefore, in the calculation of $L_s(p, n)$ in Equations 4.14 and 4.15, instead of finding the minimum of $M_s(p, q, n)$ from p_{min} to p_{max} servers, we find the minimum from β to p_{max} servers. The amount of computation is reduced because fewer combinations of p and n are needed.

Similar to Equations 4.14 and 4.15, the estimated cost for Heuristic Algorithm 4 (denoted by $L'_s(p,n)$) for the case $s < d-1$ can be written as follows:

Case 1: $s < u$

$$L'_s(p,n) = \begin{cases} \min_{q=\beta,\ldots,p_{max}} M_s(p,q,n) & \text{if } n < g_s(p_{max}), \\ M_s(p,p_{max},n) & \text{if } n \geq g_s(p_{max}). \end{cases} \tag{4.18}$$

Case 2: $s \geq u$

$$L'_s(p,n) = \begin{cases} \min_{q=\beta,\ldots,w_s} M_s(p,q,n) & \text{if } n < g_s(p_{max}), \\ M_s(p,w_s,n) & \text{if } n \geq g_s(p_{max}). \end{cases} \tag{4.19}$$

For Heuristic Algorithm 4, let l'_s be the number of servers that would yield the lowest estimated cost if this number of servers is used from time s to $s+1$. As discussed in Section 4.3.3, the key step of our cost-aware deployment heuristic algorithm is to determine l'_s for $s = 1,\ldots,d-1$. When $s = d-1$, $l'_{d-1} = l_{d-1}$ which is given by Equation 4.9. For $s < d-1$, l'_s can be obtained by a straightforward modification of Equations 4.16 and 4.17, i.e.,

For $s < u$,

$$l'_s = \begin{cases} q^* & \text{if } n < g_s(p_{max}), \\ p_{max} & \text{if } n \geq g_s(p_{max}) \end{cases} \tag{4.20}$$

where $M_s(p,q^*,n) = \min_{q=\beta,\ldots,p_{max}} M_s(p,q,n)$.

For $s \geq u$,

$$l'_s = \begin{cases} q^* & \text{if } n < g_s(p_{max}), \\ w_s & \text{if } n \geq g_s(p_{max}) \end{cases} \tag{4.21}$$

where $M_s(p, q^*, n) = \min_{q=\beta,\ldots,w_s} M_s(p, q, n)$.

Once l'_s has been determined, the decision to add or remove servers is made as follows.

1. Add $l'_s - p_s$ servers if $l'_s > p_s$
2. Remove $p_s - l'_s$ servers if $l'_s < p_s$
3. Otherwise: no change

This is shown in Algorithm 5.

Algorithm 5 Heuristic Algorithm 4: Cost-aware deployment heuristic algorithm with modified L_s

```
Ensure: The number of servers to add or remove. Positive return value means add,
    negative means remove, 0 means no change.
1: At decision point $s$, use the measured values of $p_s$ and $n_s$ (denoted by $p$ and $n$) as
   input to determine $l'_s$ from Equations 4.9, 4.20 and 4.21.
2: return $l'_s - p$
```

4.3.6 Heuristic Algorithm 5: Cost-aware deployment heuristic algorithm with modified M_s

Consider the computation of $M_s(p, q, n)$. This modified version of Heuristic Algorithm 3 is based on the following observation. In $M_s(p, q, n)$ (Equation 4.13), since r_2 is far shorter than a decision interval, M^* may not be significant. Recall that we need to try different values of p when we estimate L_s, and p is only used in finding M^*. If we set M^* to be zero, the complexity of computing L_s can be reduced from $O(dJG)$ given in to $O(dG)$.

We define the new M_s function where M^* is zero as $M'_s(p, q, n)$.

$$M'_s(p, q, n) = q \times \int_s^{s+1} c_t \, dt + \sum_{m=0}^{\infty} k_s(q, n, m) \times L_{s+1}(q, m) \tag{4.22}$$

Similar to Equations 4.14 and 4.15, the estimated cost for Heuristic Algorithm 5(denoted by $L''_s(p, n)$) for the case $s < d - 1$ can be written as follows:

Case 1: $s < u$

$$L''_s(p, n) = \begin{cases} \min_{q=p_{min},\ldots,p_{max}} M'_s(p, q, n) & \text{if } n < g_s(p_{max}), \\ M'_s(p, p_{max}, n) & \text{if } n \geq g_s(p_{max}). \end{cases} \tag{4.23}$$

Case 2: $s \geq u$

$$L''_s(p, n) = \begin{cases} \min_{q=p_{min},\ldots,w_s} M'_s(p, q, n) & \text{if } n < g_s(p_{max}), \\ M'_s(p, w_s, n) & \text{if } n \geq g_s(p_{max}). \end{cases} \tag{4.24}$$

For Heuristic Algorithm 5, let l''_s be the number of servers that would yield the lowest estimated cost if this number of servers is used from time s to $s + 1$. When $s = d - 1$, $l''_{d-1} = l_{d-1}$ which is given by Equation 4.9. For $s < d - 1$, l''_s can be obtained by a straightforward modification of Equations 4.16 and 4.17, i.e., For $s < u$,

$$l''_s = \begin{cases} q^* & \text{if } n < g_s(p_{max}), \\ p_{max} & \text{if } n \geq g_s(p_{max}) \end{cases} \tag{4.25}$$

where $M'_s(p, q^*, n) = \min_{q=p_{min},\ldots,p_{max}} M'_s(p, q, n)$.

For $s \geq u$,

$$l;'_s = \begin{cases} q^* & \text{if } n < g_s(p_{max}), \\ w_s & \text{if } n \geq g_s(p_{max}) \end{cases} \tag{4.26}$$

where $M'_s(p, q^*, n) = \min_{q=p_{min},\ldots,w_s} M'_s(p, q, n)$.

Once l''_s has been determined, the decision to add or remove servers is made as follows.

1. Add $l''_s - p_s$ servers if $l''_s > p_s$
2. Remove $p_s - l''_s$ servers if $l''_s < p_s$
3. Otherwise: no change

This is shown in Algorithm 6.

Algorithm 6 Heuristic Algorithm 5: Cost-aware deployment heuristic algorithm with modified M_s

```
Ensure: The number of servers to add or remove. Positive return value means add,
    negative means remove, 0 means no change.
1: At decision point $s$, use the measured values of $p_s$ and $n_s$ (denoted by $p$ and $n$) as
   input to determine $l''_s$ from Equations 4.9, 4.25 and 4.26.
2: return $l''_s - p$
```

Chapter 5

Performance evaluation

In this chapter, the merit of the five heuristic resource provisioning algorithms developed in the last chapter is evaluated by simulation. Our evaluation is based on the total cost of executing submitted batch jobs in a processing period, as defined in Equation 4.1.

5.1 Job processing period

In our simulation study, the job processing period under consideration is for a 24-hour day (see Figure 5.1). Decision points are 15 minutes apart. The time unit is therefore 15 minutes. The job processing period under consideration is for a 24-hour day. This period starts at 8am (time 0). The submission deadline is set to 12 midnight, i.e., $u = 64$. The completion deadline is at 7am the next morning, i.e., $d = 92$.

5.1.1 Arrival time profile

During the batch job submission period, the job arrival rate is time dependent. More jobs arrive in the middle of the submission period (time 32) and fewer jobs arrive at the

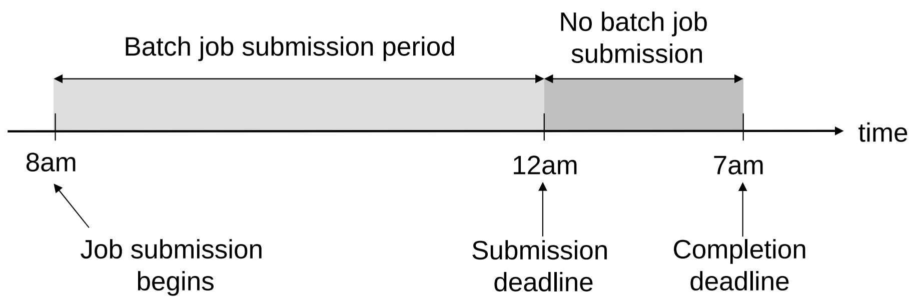

Figure 5.1: Job processing period used in experiments

two ends (time 0 and 64). Our approach to generate job arrivals that fit this profile is as follows.

We first define a quadratic function $a(x)$ given by:

$$a(x) = 2.0 - 1.04167 \times 10^{-4}x + 1.80845 \times 10^{-9}x^2 \tag{5.1}$$

$a(x)$ is obtained by linear least squares curve fitting, given the following data points: $a(x)$ = 2, 0.5 and 2 when x is 0, 28,800 and 57,600, respectively[1]. This function is depicted in Figure 5.2. It represents smaller interarrival times at x = 28,800 and larger interarrival times at the two ends of the submission period (x = 0 and x = 57,600). $a(x)$ will be used to generate job interarrival times. The algorithm is as follows:

1. Let x be the last generated arrival time, which is initialized to 0.
2. A random variate z for the exponential distribution with mean equals to 480 seconds is generated.
3. Arrival time of next job is obtained as $z \times a(x) + x$.

[1]28,800 and 57,600 refer to time 32 ($u/2$) and 64 (u) respectively.

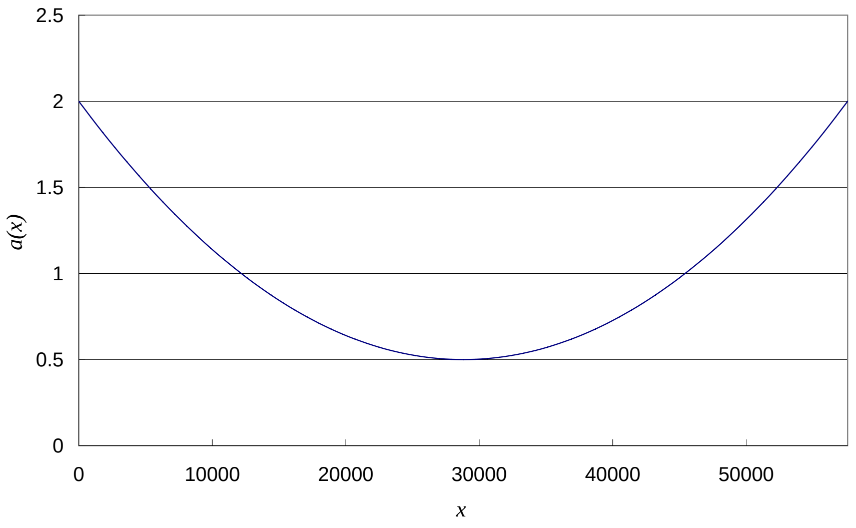

Figure 5.2: $a(x)$ from $x = 0$ to $x = 57,600$

4. x is incremented by the $z \times a(x)$.

5. Steps 2 to 4 are repeated until the generated arrival time (given by x) is larger than the submission deadline at 57,600 seconds. This last arrival is discarded and no more arrivals will be generated from this point on.

All the arrival times are stored in the job arrival profile. As an example, we generated 1,000 streams of job arrivals and collected data for the histogram of the job arrival times (measured from time 0) for each stream. The average, over all streams, of the collected data for these histograms is shown in Figure 5.3. For this example, the mean interarrival time is 394.17 seconds and its standard deviation is 33.69 seconds. The mean number of arrivals over the job processing period is 145.27 with standard deviation of 12.17.

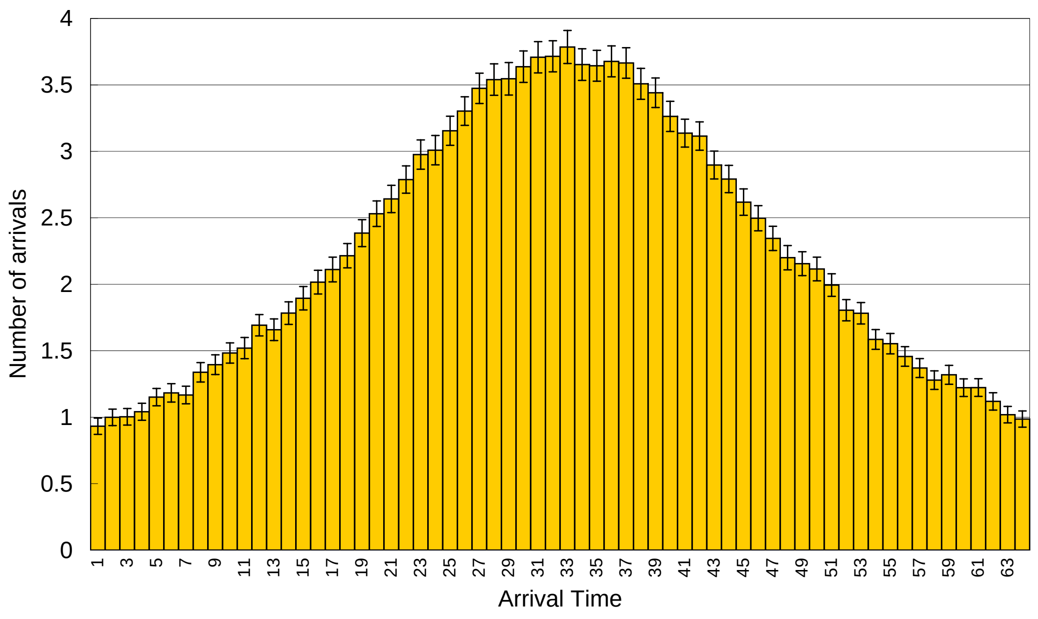

Figure 5.3: Number of arrivals at different arrival times

5.1.2 Other assumptions

The job service times are generated based on an exponential distribution with mean equal to 1,200 seconds (or 20 minutes). We assume that first-come-first-served (FCFS) scheduling is used for the cluster. The reason for using FCFS is convenience in simulation. Other scheduling disciplines can also be used with our resource provisioning algorithms. In our experiments, $p_{min} = 1$ and the time between resource provisioning decisions is 900 seconds (or 15 minutes). $p_{max} = 5$ unless stated otherwise.

5.2 Pre-computing $g_s(p)$ and $L_s(p, n)$

Our algorithms require that we pre-compute the following parameters:

- $g_s(p)$ – the largest value of n_s such that the probability of finishing all jobs by the deadline is larger than or equal to $y\%$, where $y = 99.99$ and p servers are used from time s to d, for $s = 0$ to d and $p = p_{min}$ to p_{max}

- $L_s(p, n)$ – the estimated cost from s to d given that the values of n_s and p_s at decision point s are n and p respectively, for $s = 1$ to $d - 1$ and $p = p_{min}$ to p_{max}. Note that to get $L_s(p, n)$, we need to first compute $k_s(q, n, m)$ – the probability that $n_{s+1} = m$ given that $n_s = n$ and q servers are used from s to $s + 1$, for $s = 0$ to d, $q = p_{min}$ to p_{max}, $n = 0$ to R_s, and $m = 0$ to infinity.

Methods to compute these two sets of parameters have been discussed in Sections 4.2.3 and 4.3.6, respectively.

5.3 Cost functions

Cost functions are used to model certain periods of time which we want to discourage the use of servers by the batch job cluster. Since we don't limit the shape of the cost function in the design of our heuristic algorithms, we therefore compare their performance with respect to different types of cost functions to see how well they can adapt. Although the cost functions chosen are artificial, they can provide insights in the performance of our heuristics. The cost functions under consideration are:

1. Uniform – the cost at time t c_t is independent of t.
2. Linearly increasing – c_t is an increasing function of t.
3. Linearly decreasing – c_t is a decreasing function of t.
4. Quadratic-1 – c_t is a quadratic function of t with a lower cost in the middle of the job processing period.

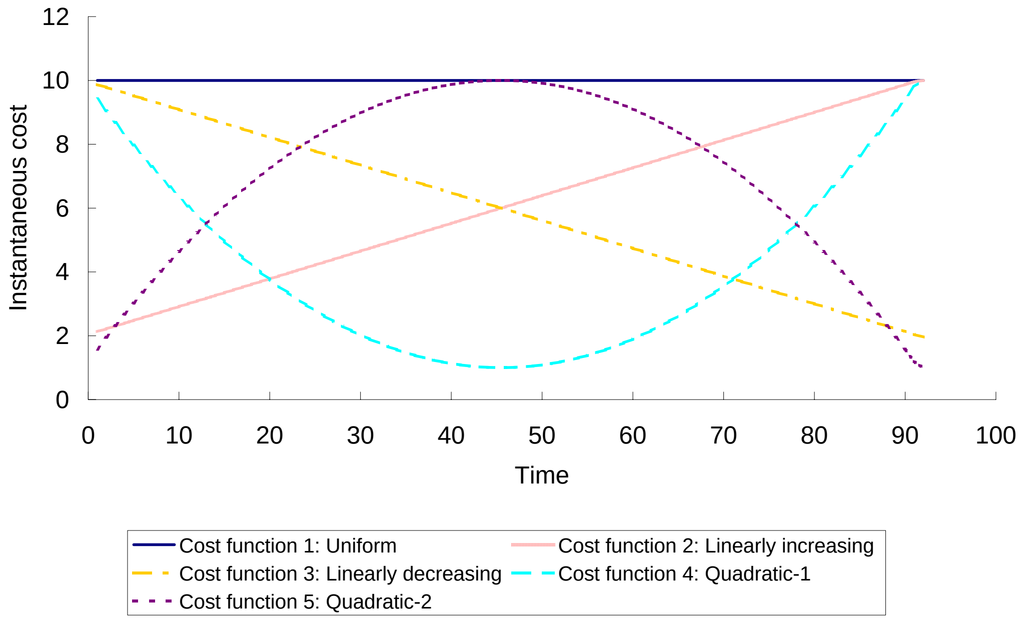

Figure 5.4: Cost functions under consideration

5. Quadratic-2 – c_t is a quadratic function of t with a higher cost in the middle of the job processing period.

The cost functions are illustrated in Figure 5.4.

5.4 Results and discussions

Simulation experiments have been performed to evaluate the merit of the five heuristic algorithms. For each experiment, we simulate the events and activities within a job processing period 1,000 times. The system is empty initially and the number of servers deployed at time zero is p_{min}. For each simulation run, results for the following performance measures are collected.

1. Number of servers used at each decision point, after a decision has been made.

2. Total cost, as defined in Equation 4.1.

3. Total number of server deployments during a job processing period.

For each experiment, 1,000 simulation runs are performed. Our evaluation of the merit of the five heuristic algorithms is based on the average value of the above performance measures over the 1,000 simulation runs.

5.4.1 Number of servers used at each decision point

Results for the number of servers used at each decision point provide insight on how the different heuristic algorithms adapt to changes in workload. These results also show that the proposed cost aware resource allocation heuristic algorithms can successfully use fewer servers during the high cost period while ensuring that all jobs meet their deadline. We observe no job missing the completion deadline for Heuristic Algorithms 1 and 2. For Heuristic Algorithms 3, 4 and 5, the percentage of simulation runs having one or more jobs missing the deadline is shown in Figure 5.5. For most of the runs missing the deadline, only one job is missed. A relatively higher value is recorded for Cost functions 3 and 5, where the cost is decreasing at the end. It indicates that the algorithms tries to delay processing jobs to the end and results in a higher chance of missing job deadline.

As mentioned previously, we use the mean value of the number of servers used at each decision point over 1,000 simulation runs. The 95% confidence intervals have also been computed. They are very small (less than 0.1) and are therefore not shown when we present our results.

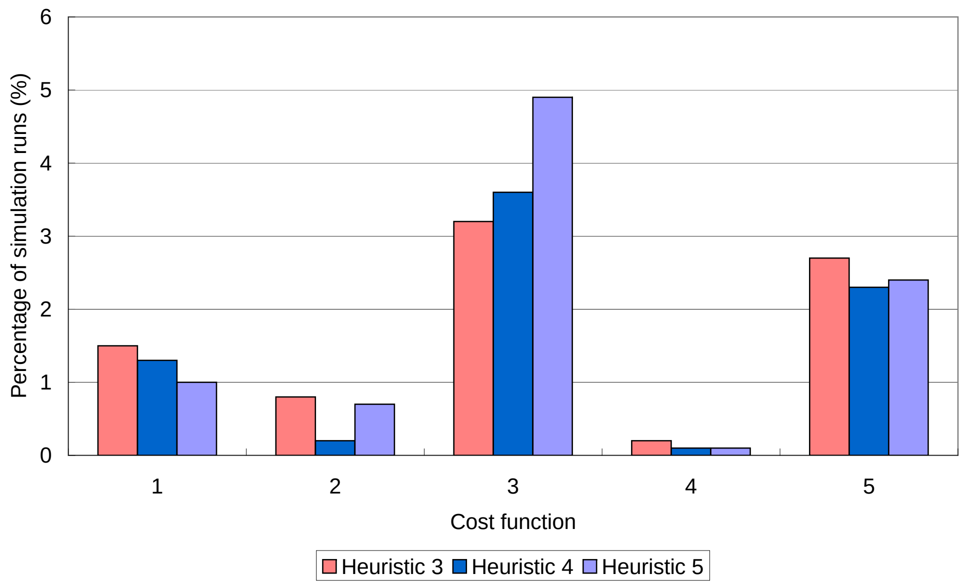

Figure 5.5: Percentage of simulation runs having job(s) missing the completion deadline.

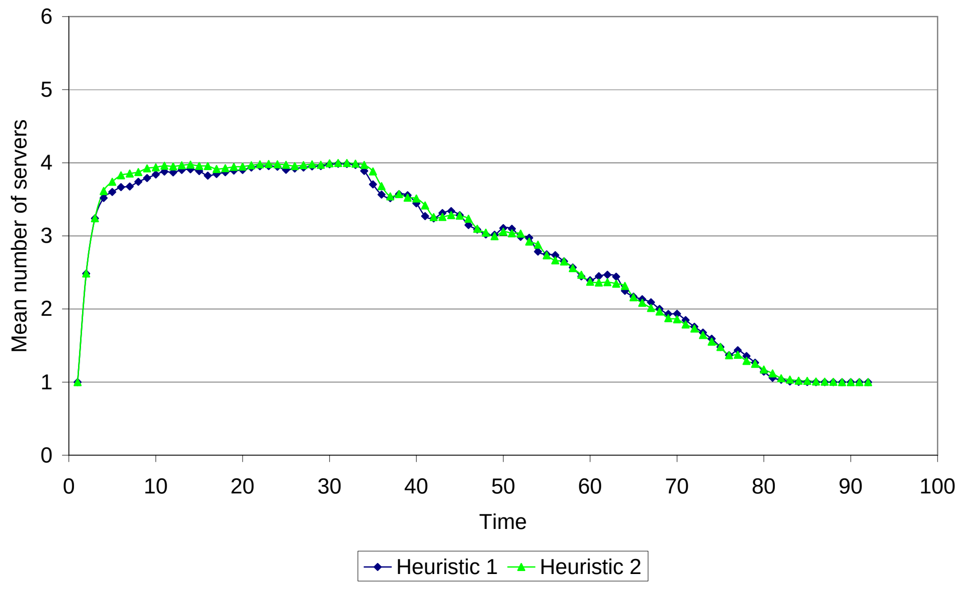

Figure 5.6: Mean number of servers used at each decision point: Heuristic Algorithms 1 and 2

Simple heuristic algorithms – Heuristic Algorithms 1 and 2

We first consider Heuristic Algorithms 1 and 2, described in Algorithms 1 and 2, respectively. The cost function is not taken into consideration in the design of these heuristic algorithms. As a result, the behaviors of these heuristic algorithms are not affected by the cost function.

The number of servers used by Heuristic Algorithms 1 and 2 as a function of time is plotted in Figure 5.6. We observe that these two heuristic algorithms deploy enough servers at the beginning to ensure that there is sufficient confidence of meeting the deadline for all jobs. Then, they gradually reduce the number of servers used until all jobs are finished. We also observe that the two heuristic algorithms have very similar behavior in terms of the number of servers used at each decision point.

Cost aware resource allocation – Heuristic Algorithms 3, 4 and 5

We next consider Heuristic Algorithms 3, 4 and 5, described in Algorithms 3, 5 and 6, respectively. Their behaviors with respect to the cost functions under consideration are discussed below.

Cost function 1: Uniform. The behavior of Heuristic Algorithms 3, 4 and 5 for the Uniform cost function is shown in Figure 5.7. We observe that the number of servers used for Heuristic 5 fluctuates a lot. It shows that although M^* is not significant in Heuristics 3 and 4, it can stabilize the number of servers in the system.

Cost function 2: Linearly increasing. The results shown in Figure 5.8 indicate that the heuristic algorithms are aware of the fact that cost is an increasing function of time and use more servers at the beginning. Although the cost is the lowest at the beginning, the heuristic algorithms do not sharply increase the number of servers. They simply ensure that servers do not become idle. This is confirmed by the observation that p_s, as shown

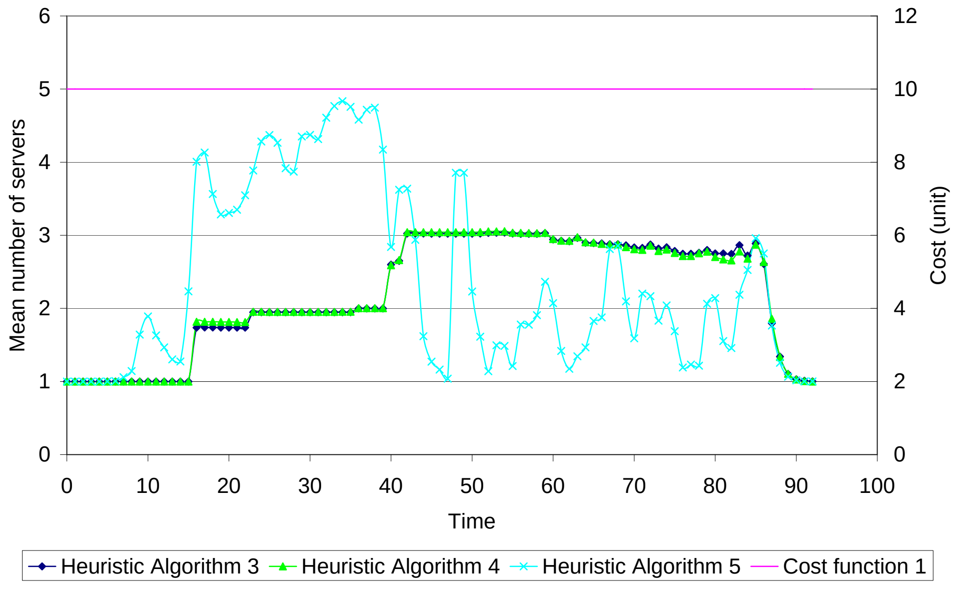

Figure 5.7: Mean number of servers used at each decision point: Cost function 1

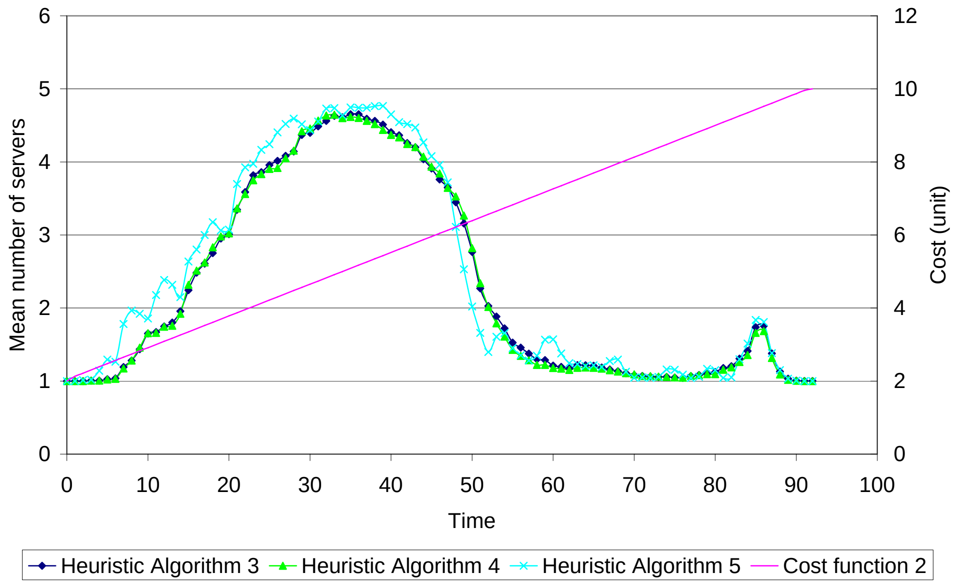

Figure 5.8: Mean number of servers used at each decision point: Cost function 2

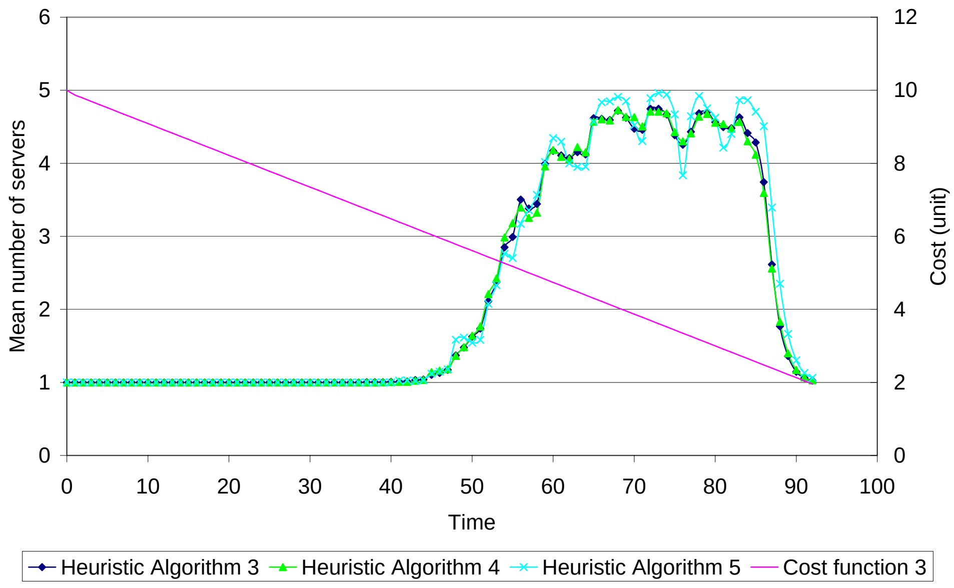

Figure 5.9: Mean number of servers used at each decision point: Cost function 3

in Figure 5.8, is smaller (but not much smaller) than the corresponding n_s, as shown in Figures 5.12, 5.13 and 5.14. After the peak at around time 35, the number of servers begins to drop until it is close to $p_{min} = 1$.

There is a small peak at time 85. This can be explained as follows. The value of $g_s(p_{max})$, as shown in Figures 5.12, 5.13 and 5.14, tends to be a decreasing function of s after the submission deadline (at time 64) has been reached. There is a sudden drop of $g_s(p_{max})$ from 9 jobs at $s = 84$ to 3 jobs at $s = 85$. For Heuristic Algorithm 3, the mean number of jobs at time 85 is 0.837 with standard deviation 1.213. Similarly, for Heuristic Algorithms 4 and 5, the mean number of jobs at time 85 is 0.752 with standard deviation 1.094 and 0.905 with standard deviation 1.152 respectively. As a result, there are some simulation runs with $n_s > g_s(p_{max}) = 3$ when $s = 85$. In those cases, additional servers are deployed, which results in the small peak.

Cost function 3: Linearly decreasing. The results shown in Figure 5.9 indicate

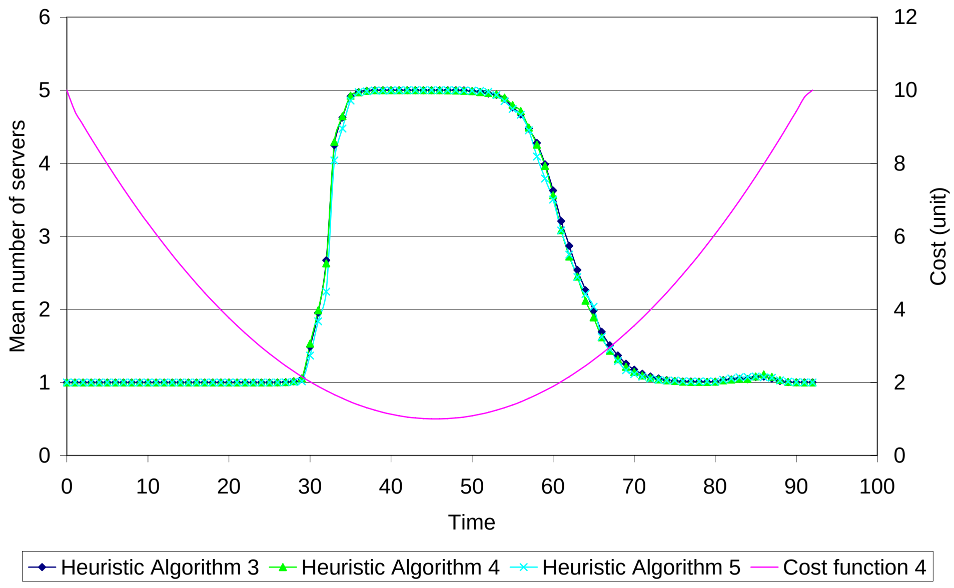

Figure 5.10: Mean number of servers used at each decision point: Cost function 4

that the heuristic algorithms avoid the high cost period at the beginning by deferring most of the server usage to the end of the job processing period. The number of servers used remains at a low level from 0s to around time 34. Then it gradually increases to $p_{max} = 5$ at time 68. This increase is due to the accumulation of jobs in the system and the number of jobs reaches $g_s(p_{max})$ at time 68. The gradual increase in the number of servers used (from time 34 to 68) can be viewed as a supporting evidence that the heuristic algorithms are attempting to balance between incurring higher cost by adding servers too early and incurring higher cost when $g_s(p_{max})$ is reached too early.

As shown in Figures 5.12, 5.13 and 5.14, n_s is very close to $g_s(p_{max})$ at the end of the job processing period. This is additional evidence that the heuristic algorithms avoid the high cost period by leaving more jobs to be processed at the end.

Cost function 4: Quadratic-1. The cost is higher at the beginning and near the completion deadline. As shown in Figure 5.10, the heuristic algorithms use more servers

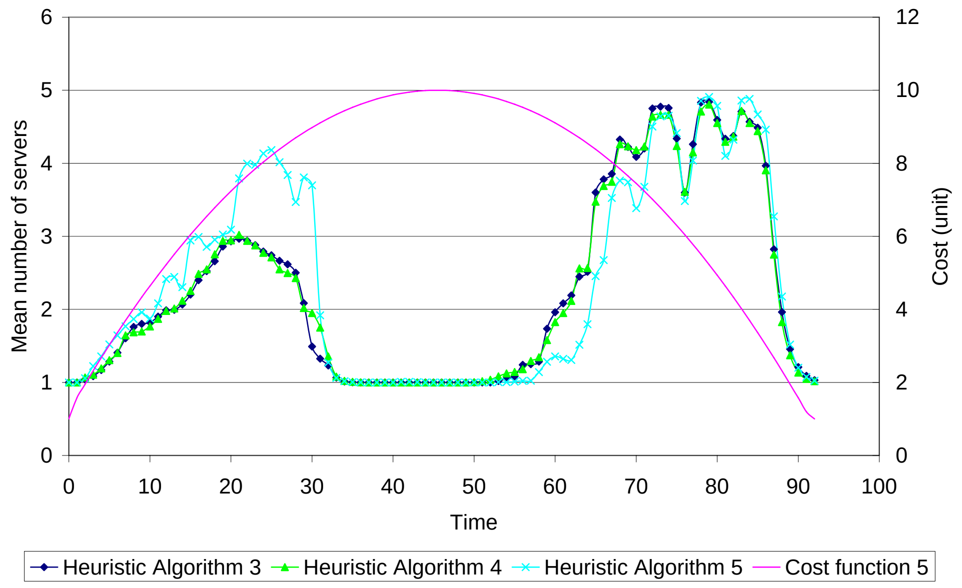

Figure 5.11: Mean number of servers used at each decision point: Cost function 5

during the low cost period in the middle. They also delay the usage of servers at the beginning, and avoid the usage of servers near the completion deadline.

Cost function 5: Quadratic-2. From Figure 5.11, the heuristic algorithms try to process as many jobs as possible from time 0 to 36 by using more servers. Then, they decide that it is desirable to remove servers during the high cost period in the middle. After 54,900s, the number of servers increases sharply because the number of jobs accumulated in the system is significant (see Figures 5.12, 5.13 and 5.14).

5.4.2 Total cost

The total cost of the five heuristic algorithms for the five cost functions considered are shown in Figure 5.15. We also include the cost of static provisioning which is based on the use of four servers throughout the job processing period. Four is the minimum number of

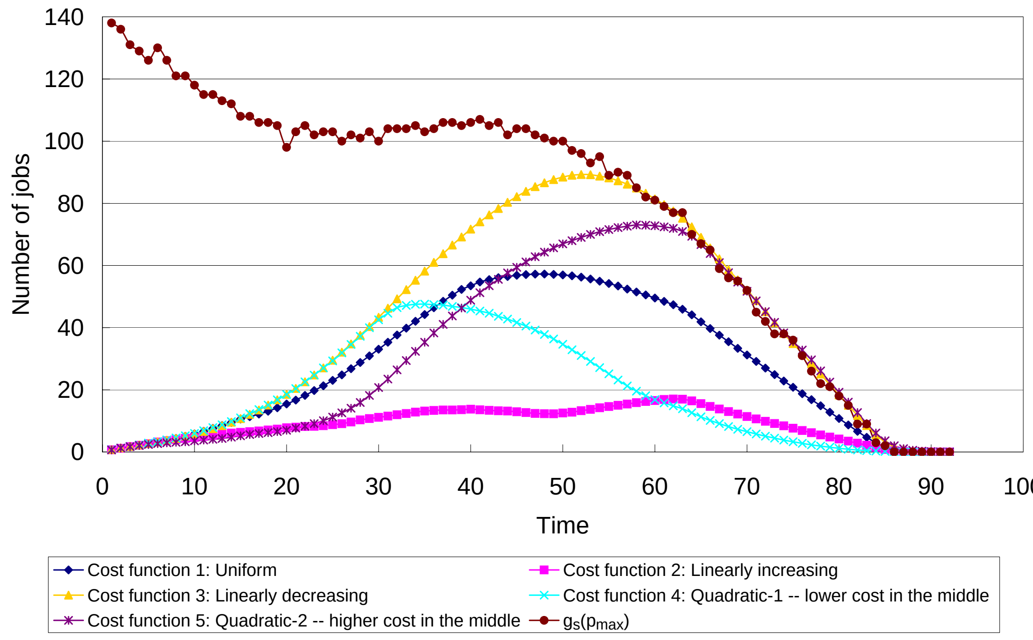

Figure 5.12: Mean number of jobs at each decision point: Heuristic Algorithm 3

servers required to provide a 99.99% probability that all jobs are finished by the deadline. The following observations are made from the results in Figure 5.15:

1. The total costs of Heuristic Algorithms 3, 4 and 5 are almost the same. Since Heuristic Algorithms 4 and 5 are more efficient with respect to computation time, they are more preferred than Heuristic Algorithm 3.

2. For the cases considered in our experiments, Heuristic Algorithms 3, 4 and 5 are able to achieve savings of about 15 - 40% and 40 - 60% in the total cost, when compared to the simple heuristic algorithms (Heuristic Algorithms 1 and 2) and static server allocation, respectively.

3. Even when a uniform cost function is used, Heuristic Algorithms 3, 4 and 5 perform better than Heuristic Algorithms 1 and 2; the saving in total cost is around 20%.

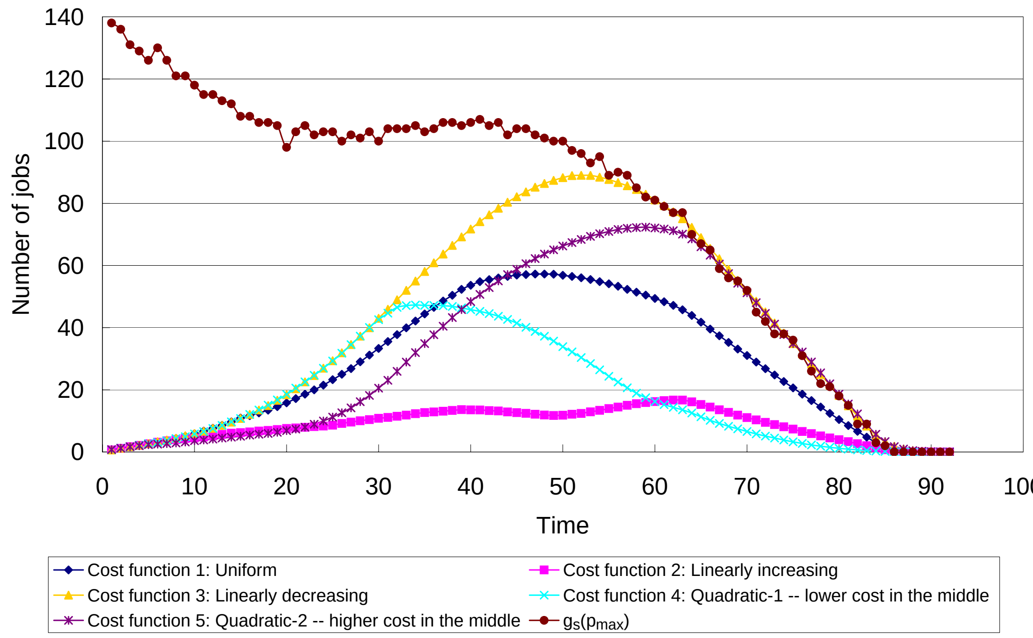

Figure 5.13: Mean number of jobs at each decision point: Heuristic Algorithm 4

5.4.3 Total number of server deployments during a job processing period

The total number of server deployments during a job processing period is a measure of how frequently servers are added to the cluster. Frequent deployment could result in excessive overhead. In Figure 5.16, we show the number of deployments for the five heuristic algorithms. We observe that Heuristic Algorithms 3 and 4 do not change the number of servers often. For Heuristic Algorithm 5, the number of deployments is more than that of Heuristic Algorithms 3 and 4 since M^* is not included in $M'_s(p, q, n)$ (Equation 4.22). For cost function 5 (Quadratic-2), server deployment activities happen twice, at the two ends of the job processing period, resulting in a larger total number of deployments than for the other cost functions. For Heuristic Algorithms 1 and 2, the delayed removal approach

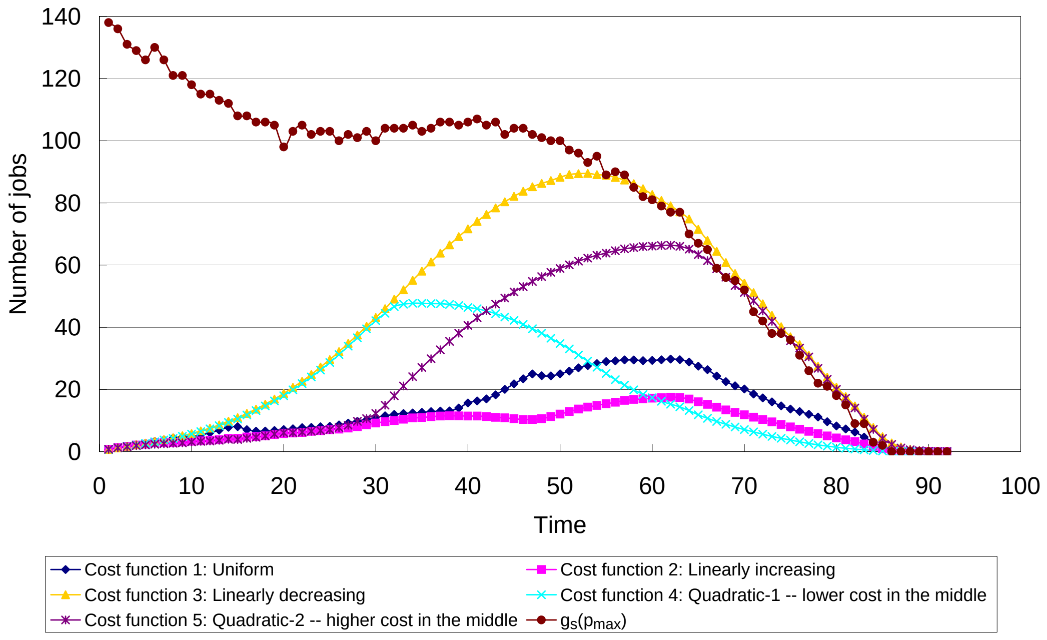

Figure 5.14: Mean number of jobs at each decision point: Heuristic Algorithm 5

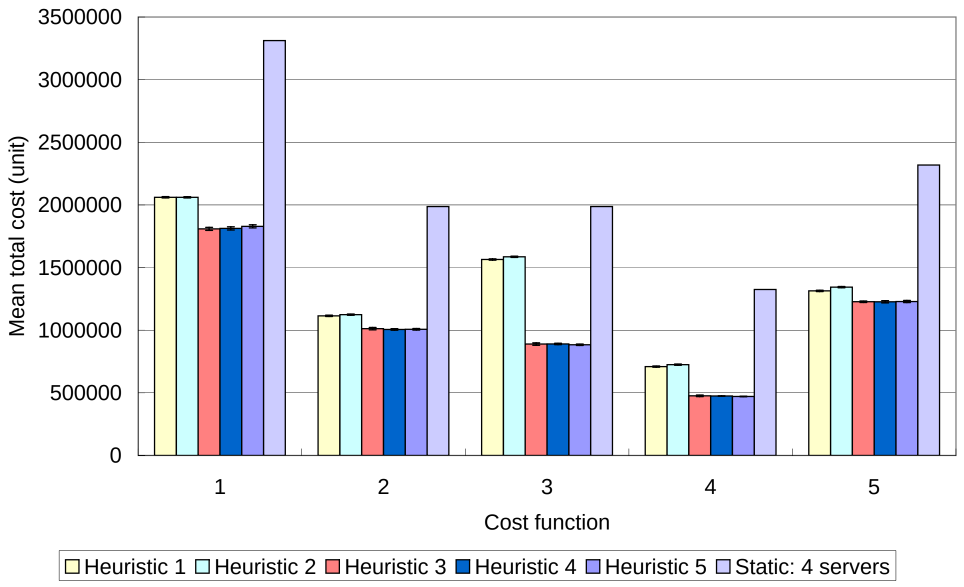

Figure 5.15: Mean total cost of different heuristic algorithms under different cost functions.

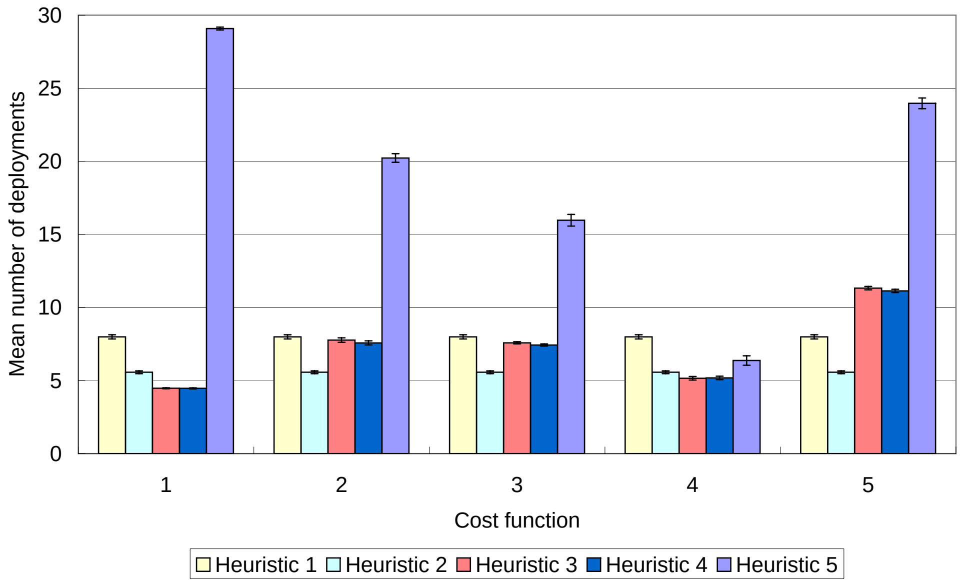

Figure 5.16: Mean number of deployments of different heuristic algorithms under different cost functions.

of Heuristic Algorithm 2 leads to a smaller total number of deployments than Heuristic Algorithm 1.

5.4.4 Scalability

In autonomic resource provisioning, allocation decisions need to be made quickly at each decision point. For Heuristics 3, 4 and 5, online computation of $L_s(p,n)$, $L'_s(p,n)$ and $L''_s(p,n)$ is not possible when $J = p_{max} - p_{min}$ and $G = \max_{s=0}^{d-1} g_s(p_{max})$ are large. We must therefore pre-compute them so that the data required for making decisions are available without much delay. To pre-compute $L_s(p,n)$, $L'_s(p,n)$ and $L''_s(p,n)$, we must first compute $g_s(p_{max})$ and $k_s(q,n,m)$. The method to compute $g_s(p_{max})$ is discussed in Sections 4.2.3 and those for $k_s(q,n,m)$, $L_s(p,n)$, $L'_s(p,n)$ and $L''_s(p,n)$ are discussed in Sections 4.3.6.

In this section, we measure the time required to compute $L_s(p,n)$, , $L'_s(p,n)$, $L''_s(p,n)$,

p_{max}	Mean number of arrivals	Mean inter-arrival time (seconds)
5	144.74	393.52
10	291.06	197.51
20	583.12	99.04
30	878.38	65.98

Table 5.1: Mean inter-arrival time and mean number of arrivals used for the scalability tests

$g_s(p_{max})$ and $k_s(q,n,m)$ for different values of p_{max}. The results will provide insight into the scalability of our heuristic algorithms. In the scalability tests, we assume that, when the p_{max} increases, the number of arrivals during a job processing period is correspondingly higher. The mean number of arrivals, over 1,000 arrival streams are shown in Table 5.1. Note that a larger number of arrivals implies a smaller mean interarrival time. The mean interarrival times are also shown in Table 5.1.

The time to complete the computation of $L_s(p,n)$, $L'_s(p,n)$, $L''_s(p,n)$, $g_s(p_{max})$ and $k_s(q,n,m)$ for $p_{max} = 5$, 10, 20 and 30 are shown in Figure 5.17. The computations are performed (using Java) on a PC equipped with an Intel 630 CPU (3GHz, Hyper-threaded, 2MB Cache, 64-bit and single core), 512MB RAM, and a Fedora Core 3 Linux operating system. We observe that the time required to compute $k_s(q,n,m)$ is significant. This is because of the large number of simulations required.

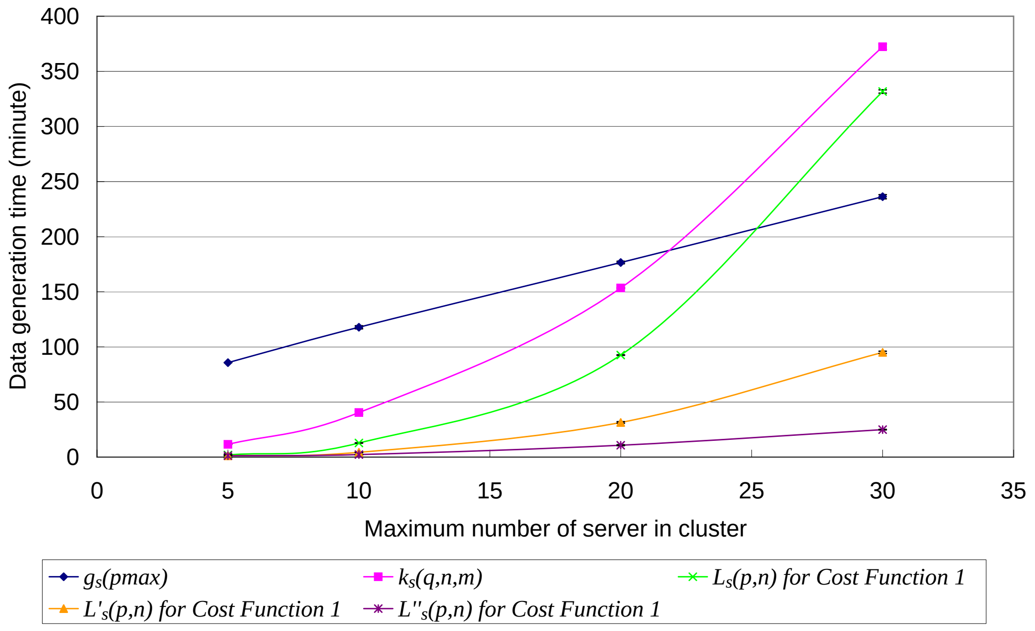

Figure 5.17: Computation time (minutes) of $g_s(p_{max})$, $k_s(q, n, m)$, $L_s(p, n)$, $L'_s(p, n)$ and $L''_s(p, n)$ values

Chapter 6

Conclusion and future work

6.1 Conclusion

In this thesis, we have developed five autonomic resource allocation heuristic algorithms for batch job clusters. In large scale server clusters, the capabilities to manage their own resources are useful in reducing over-provisioning, reducing operation cost and enhancing resource utilization. We show that our heuristic algorithms are effective with respect to provisioning of server resources in a batch job cluster. The first two heuristic algorithms (Heuristic Algorithms 1 and 2), which are not cost-aware, are adaptive to current and future workload. The other three heuristic algorithms (Heuristic Algorithms 3, 4 and 5) are cost-aware and predictive in future resource usage. They are superior to Heuristic Algorithms 1 and 2. The total costs of Heuristic Algorithms 3, 4 and 5 are almost the same. Since Heuristic Algorithms 4 and 5 is more efficient in computation, they are more preferred than Heuristic Algorithms 3. Heuristic Algorithms 4 is better than Heuristic Algorithms 5 if we want to minimize the fluctuations in number of servers used during the job processing period. However Heuristic Algorithms 4 has a higher time complexity than

Heuristic Algorithms 5.

6.2 Summary of contributions

We summarize our contributions as follows:

1. We proposed five heuristic algorithms for deciding the number of servers to use in a cluster at decision points.
2. We have considered various time-varying cost functions when making resource allocation decisions. These functions model the external factors that affect the number of servers to use in a cluster.
3. We have included in our model the overhead in changing the number of servers used in the cluster. This overhead is not considered in most of the previous work.
4. We have demonstrated that our cost aware resource allocation heuristic algorithms outperform static allocation and heuristic algorithms that are not based on cost considerations.
5. The proposed resource allocation heuristic algorithms are applicable to existing batch job clusters.

6.3 Future work

Areas for future work include the following.

1. **Other heuristics to minimize the total cost.** One possible approach is to use a linear programming formulation. At time s, we determine q_s, the number of servers

to use at time s. We assume that jobs only arrive and depart at decision points. Let m_s be the mean number of job arrivals between time $s-1$ to s. $m_s = \lambda_s$ where λ_s is the average arrival rate from time $s-1$ to s. We assume that these jobs arrive at time s. Let b_s be the number of active servers at time s. $b_s = \min\{n_s + m_s, q_s\}$. For each job at time s, the probability that this job complete service at or before time $s+1$ (under exponential service time distribution) is equal to $1 - e^{-x}$. The mean number of job completion from time s to $s+1$ is therefore $b_s(1 - e^{-x})$. We assume that these completions occur at time $s+1$. The objective function to minimize is: $q_s\overline{c}_s + q_{s+1}\overline{c}_{s+1} + \ldots + q_{d-1}\overline{c}_{d-1}$ where $\overline{c}_s = \int_s^{s+1} c_t \; dt$. The constraints are: i) $n_{j+1} = \max\{n_j + m_j - b_j(1-e^{-x}), 0\}$ for $j = s, s+1, \ldots, d-1$, and ii) $n_d = 0$, i.e., no job misses the deadline. This optimization problem is solved at every decision point. The solution provides the information needed for making provisioning decisions.

2. **Batch jobs with different deadline requirement.** Our study is based on batch jobs with a common deadline. Another scenario of interest is that each batch job may have a different completion deadline. The extension of our heuristic algorithms or the development of new heuristic algorithms for this scenario should be investigated. While deadline-based scheduling for batch job clusters have been widely studied, not much attention has been put on dynamic resource allocation of such systems.

3. **Batch jobs with dependencies.** We have assumed that batch jobs are independent in our heuristic algorithms. When jobs are not independent of each other in terms of execution order, the resource provisioning algorithm may need to consider this information when predicting future resource usage.

4. **Heterogeneous cluster.** Autonomic provisioning to a heterogeneous batch job cluster would be an interesting and challenging topic to explore. First, in addition to

when servers should be allocated (or removed), determining which servers to allocate (or remove) may not be straightforward. For example, the system needs to differentiate between the benefits of adding one fast server or two slower servers. Secondly, a server equipped with a faster CPU does not necessarily run all kinds of jobs faster than another server because of the jobs' I/O and memory requirements. The added complexity makes it more difficult to predict the job completion time.

5. **Batch job cluster with no check-pointing of jobs.** For clusters with no check-pointing, if we wish to remove a server node, we can either wait until a server node becomes idle or remove the node even when it is busy running a job. For the former approach, we need to estimate the waiting time and include it in the removal overhead. For the latter approach, CPU time is wasted, which may not be desired. Of interest is an assessment of the relative merit of these two approaches.

Bibliography

[1] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] Karen Appleby, Sameh Fakhouri, Liana Fong, Germán Goldszmidt, Michael Kalantar, Srirama Krishnakumar, Donald Pazel, John Pershing, and Benny Rochwerger. Oceano - SLA based management of a computing utility. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, pages 855–868, May 2001.

[3] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 90–101, New York, NY, USA, 2000. ACM Press.

[4] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: A new facility for resource management in server systems. In *OSDI '99: Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 45–58, Berkeley, CA, USA, 1999. USENIX Association.

[5] Galen Barbose, Charles Goldman, and Bernie Neenan. Survey of utility experience with real time pricing, December 2004. Lawrence Berkeley National Laboratory, LBNL-54238. `http://eetd.lbl.gov/ea/ems/reports/54238.pdf`.

[6] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. The case for power management in web servers. pages 261–289, 2002.

[7] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pages 103–116, New York, NY, USA, 2001. ACM Press.

[8] Jeffrey S. Chase, David E. Irwin, Laura E. Grit, Justin D. Moore, and Sara E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, pages 90–100, Washington, DC, USA, 2003. IEEE Computer Society.

[9] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing server energy and operational costs in hosting centers. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 303–314, New York, NY, USA, 2005. ACM Press.

[10] Condor Team, A Resource Manager for High Throughput Computing, Software Project, The University of Wisconsin, `http://www.cs.wisc.edu/condor`.

[11] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems.* John Wiley & Sons, 2004.

[12] HP Utility Data Center, Hewlett-Packard. `http://h71028.www7.hp.com/enterprise/cache/259007-0-0-225-121.html?jumpid=reg_R1002_USEN`.

[13] IBM Tivoli Intelligent Orchestrator, IBM. `http://www-306.ibm.com/software/tivoli/products/intell-orch/`.

[14] Intel Preboot Execution Environment (PXE). `http://www.intel.com/labs/manage/wfm/tools/pxesdk20/index.htm`.

[15] R. Levy, J. Nagarajarao, G. Pacifici, A. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. In *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management*, pages 247–261, March 2003.

[16] Marin Litoiu, Murray Woodside, and Tao Zheng. Hierarchical model-based autonomic control of software systems. In *DEAS '05: Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software*, pages 1–7, New York, NY, USA, 2005. ACM Press.

[17] Zhen Liu, Mark S. Squillante, and Joel L. Wolf. On maximizing service-level-agreement profits. *SIGMETRICS Perform. Eval. Rev.*, 29(3):43–44, 2001.

[18] LSF Administrator's Guide, Version 4.1, Platform Computing Corporation, February, 2001.

[19] Edson Manoel, Sara Carlstead Brumfield, Kim Converse, Mark DuMont, Leonard Hand, Gordon Lilly, Morten Moeller, Adam Nemati, and Al Waisanen. *Provisioning On Demand: Introducing IBM Tivoli Intelligent ThinkDynamic Orchestrator.* IBM International Technical Support Organization, December 2003. `http://www.redbooks.ibm.com`.

[20] Daniel A. Menascé, Mohamed N. Bennani, and Honglei Ruan. On the use of online analytic performance models, in self-managing and self-organizing computer systems. In Özalp Babaoglu, Márk Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi, Aad P. A. van Moorsel, and Maarten van Steen, editors, *Self-star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2005.

[21] Justin Moore, David Irwin, Laura Grit, Sara Sprenkle, and Jeff Chase. Managing mixed-use clusters with cluster-on-demand. Technical report, Duke University, Department of Computer Science, November 2002.

[22] Open-PBS Team, A Batching Queuing System, Software Project, Altair Grid Technologies, LLC, `http://www.openpbs.org`.

[23] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. QoS-driven server migration for Internet data centers. In *Proceedings of ACM/IEEE International Workshop on Quality of Service (IWQoS)*, pages 3–12, Miami Beach, FL, USA, May 2002.

[24] J. Rolia, S. Singhal, and R. Friedrich. Adaptive internet data centers. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR'00)*, pages 3–12, L'Aquila, Italy, July 2000.

[25] Kai Shen, Hong Tang, Tao Yang, and Lingkun Chu. Integrated resource management for cluster-based internet services. *SIGOPS Oper. Syst. Rev.*, 36(SI):225–238, 2002.

[26] Gurmeet Singh, Carl Kesselman, and Ewa Deelman. Performance impact of resource provisioning on workflows. Technical report, University of Southern California, Computer Science Department, 2005.

[27] Gokul Soundararajan and Cristiana Amza. Online data migration for autonomic provisioning of databases in dynamic content web servers. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 268–282. IBM Press, 2005.

[28] Sun GridEngine. `http://gridengine.sunsource.net/`.

[29] L.G. Alex Sung, Johnny W. Wong, Marin Litoiu, and Gabriel Iszlai. Dynamic provisioning of processor nodes in a grid computing environment. In *Proceedings of the Second International Workshop on Smart Grid Technologies*, Dublin, Ireland, June 2006. To appear.

[30] Bhuvan Urgaonkar and Prashant J. Shenoy. Sharc: Managing CPU and network bandwidth in shared clusters. *IEEE Trans. Parallel Distrib. Syst.*, 15(1):2–17, 2004.

[31] Qi Zhang, Evgenia Smirni, and Gianfranco Ciardo. Profit-driven service differentiation in transient environments. In *11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03)*, pages 230–233, Orlando, FL, USA, October 2003.

[32] Huican Zhu, Hong Tang, and Tao Yang. Demand-driven service differentiation in cluster-based network servers. In *Proceedings of IEEE INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 679–688, Anchorage, AK, USA, April 2001.