European Journal of Operational Research 210 (2011) 344–357

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

Cost-based decision-making in middleware virtualization environments

Kaushik Dutta *, Debra VanderMeer

College of Business, Florida International University, Miami, FL, United States

ARTICLE INFO

Article history:
Received 24 April 2009
Accepted 2 October 2010
Available online 29 October 2010

Keywords:
Computing science
Virtualization
Resource assignment
System design

ABSTRACT

Middleware virtualization refers to the process of running applications on a set of resources (e.g., databases, application servers, other transactional service resources) such that the resource-to-application binding can be changed dynamically on the basis of applications' resource requirements. Although virtualization is a rapidly growing area, little formal academic or industrial research provides guidelines for cost-optimal allocation strategies. In this work, we study this problem formally. We identify the problem and describe why existing schemes cannot be applied directly. We then formulate a mathematical model describing the business costs of virtualization. We develop runtime models of virtualization decision-making paradigms. We describe the cost implications of various runtime models and consider the cost effects of different managerial decisions and business factors, such as budget changes and changes in demand. Our results yield useful insights for managers in making virtualization decisions.

1. Introduction

Virtualization is a growing part of the overall information technology market. The 451 Group predicts that the virtualization market overall will grow from $2.2B in 2008 to nearly $11B in 2013 (Kusnetzky, 2009). A particular type of virtualization, *middleware virtualization*, also known as *Platform as a Service*, allows an application to run on a set of middleware resources such that the resource-to-application binding can be changed dynamically on the basis of each application's resource requirements. In other words, middleware resources can be deployed and un-deployed to support an application's workload needs as demand rises and falls. Here, the term *middleware* refers to application middleware software platforms (e.g., WebSphere (IBM Inc., 2006b), IIS (Microsoft Inc., 2009), WebLogic (BEA Systems Inc., 2004), others) that provide generic application services (e.g., database connection management, thread pool management, naming and directory services, and other application support services), as well as other application support systems, e.g., database servers or transaction servers. Applications written to run on application middleware platforms need only implement their specific business logic to take advantage of the generic services available from the middleware, i.e., they need not re-implement the same generic functions themselves.

In a middleware virtualization scenario, *middleware stacks* consisting of middleware software and the operating system and hardware resources supporting it (e.g., as depicted in Fig. 1), can be provisioned as needed to support any application written for the platform (i.e., Java EE platforms can support Java-based applications written to the Java EE standard, while IIS can support .NET-based applications). Indeed, by installing multiple middleware frameworks on each middleware stack instance, it is possible for a middleware stack to support any application simply by starting the appropriate middleware software and deploying the application.

Middleware virtualization differs significantly from server virtualization, which allows multiple guest operating systems to run on a single host machine, accessing a common set of hardware resources (with the attendant additional delays associated with the extra layer of indirection imposed by the guest operating system). In contrast, middleware virtualization technologies allow multiple applications to share a pool of middleware stacks. This enables data center managers to dynamically allocate and deallocate application resources without interrupting the runtime processing of an application.

Typically, several application clusters supported by middleware stacks reside within a data center, where each application cluster supports a single application. Instead of permanently sizing application clusters for peak loads, managers should be able to reallocate resources such as application servers, database servers and storage servers in response to the current demand for each application. Because different applications are unlikely to experience peak demand simultaneously, managers can save money by reducing the total number of units deployed and moving idle resource units from cluster to cluster as demand dictates. If total demand exceeds the available resources, applications can be prioritized to ensure that critical systems do not starve. Then, if increased

* Corresponding author. Tel.: +1 305 348 3302.
E-mail addresses: kaushik.dutta@fiu.edu (K. Dutta), debra.vandermeer@fiu.edu (D. VanderMeer).

0377-2217/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2010.10.006

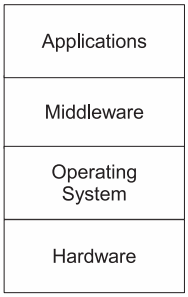

Fig. 1. Middleware virtualization resource stack.

demand persists, administrators can boost their capacity by adding new resource units to the existing infrastructure pool.

Consider the case of a major credit card company (the name has been withheld in order to honor confidentiality agreements) in New York that has several applications running in multiple data centers. The merchant credit (MC) and credit card bill check-payment (CCB) applications are two important applications in the company's data centers. These applications use various resources like database servers, remote web services, security services, application servers and storage services. A number of instances of each of these resources are located in multiple data centers located in four geographically separated cities in the United States. At any given time the number of applications running in these data centers ranges between 90 and 120. Over the course of a day, the MC application reaches its peak load between 8 a.m. and 12 noon, when merchant activity is heavy. The CCB application reaches peak loads between 4 p.m. and 8 p.m., when large numbers of customers check and pay credit card bills. During peak loads, both applications combined use approximately 85% of available physical resources. During non-peak periods, these applications use on average 20–30% of the resources. Clearly, there is often a significant under-utilization of resources. This scenario is typical of IT data centers.

In order to better utilize resources, what is needed is a mechanism to allow the MC and CCB applications to temporarily expand within a common set of resources during their peak loads. Such a mechanism would increase the average utilization of resources and enable the company to run a larger set of applications on the same set of resources. *Middleware virtualization technologies present the possibility of achieving this goal.* In this context, the application set must be mapped to a set of specific middleware stack resources. Here, the following question arises: *on what basis should we decide which application to assign to which middleware stack?* For many enterprises, the main motivation for deploying middleware virtualization technology is to manage costs. Thus, the problem of resource-to-application allocation needs to be tackled in both a cost-effective and QoS-friendly manner. *The key problem businesses face in using virtualization technology is how the virtualized resources can be utilized to promote cost and business priorities* (Business Wire, 2007). There is a gap between technical know-how and the achievement of business goals that has not been addressed in detail in the literature – while managers would like to ensure minimum-cost resource-to-application allocations, they currently have no way to determine the cost implications of allocations due to the complexity of the decision.

This complexity is based on a number of factors, starting with the difficulty of optimizing over a wide variety of applications and resources. This problem is exacerbated by the complexity introduced by multiple geographically separate data centers, where each location has a different cost profile, based on a variety of factors. We cite two examples of cost-differentiating factors, the costs of powering data centers and the cost of human IT resources, below.

Power usage is a significant cost differentiator – data centers are notorious power consumers, both to run the servers as well as to cool the server rooms. In fact, recent research indicates that the cost of powering and cooling a data center actually exceeds the cost of the IT equipment the data center houses (Belady, 2007). Further, power costs differ substantially across regional areas – in January 2010, the average cost per kilowatt-hour for commercial use was $0.15 in the state of New York, but only $0.074 in Oregon and $0.063 in North Dakota (US Energy Information Administration, 2010). Such disparities in power costs have led many data center operators, e.g., Google and Microsoft, to consider locations with low-cost power sources, e.g., based on hydrodynamic (Scheier, 2007) or geothermal (Hancock, 2009) sources.

The cost of human IT resources also varies significantly on a regional basis. This is clearly demonstrated by the fact that most major IT salary surveys report average on a regional basis – ComputerWorld reports average salaries on a multi-state regional basis (ComputerWorld, 2009), while salary.com provides wage estimates by metropolitan regional area (Salary.com, 2010). For example, based on salary.com data, a systems administrator in New York costs 14% more than one in Portland, Oregon.

These difficulties are compounded by the problem of varied incentives – application users want fast response times regardless of cost, application owners want the fast response times without spending too much money, and data center managers seek to reduce the cost of running all applications within their service agreements, regardless of ownership. These competing incentives raise the question – whose incentives should be paramount in the resource-to-application mapping decision?

These incentives can play out in a variety of managerial scenarios, with slightly differing implications based on how application owners are charged for their applications. Essentially, application owners are responsible for the final cost, whereas the resource managers are responsible for managing costs.

In the first scenario, in-house IT staff make allocation decisions, and IT operations costs are not tied back to application resource usage. IT staff are incentivized to reduce costs across the board to minimize budget requirements for upper management. Application owners want maximum performance for their applications, without regard to cost.

In the second scenario, in-house IT staff make allocation decisions, but with a charge-back policy (McKinnon and Kallman, 1987) in place to tie application usage costs back to the application owners. IT staff is still incentivized to reduce costs across the board, but application owners are now incentivized to minimize cost of owned applications, regardless of cost or performance impacts on other applications.

In the third and final scenario, applications are hosted in outsourced data centers (e.g., perhaps Amazon's Elastic Compute Cloud (Amazon Web Services, 2010)), where application owners are charged based on actual usage (Stone and Vance, 2010). Data center managers have an incentive to maximize profit, while application owners have an incentive to minimize their own costs.

In all three scenarios, application users will tolerate little in the way of delays, regardless of application workloads. However, these stakeholders have little or no control over the main parameters that drive the allocation decision – the cost of resources and allocation events, application budget limits, and demand for applications – and no way of understanding the interactions between these parameters without assistance. In this work, we take a first step toward helping stakeholders understand implications of virtualization, and develop a set of insights to help them understand what they can expect as the values of these parameters change.

In such complex scenarios, it is virtually impossible to make resource allocation decisions without a formal framework, thus motivating this research. We attempt to address this gap, providing

managers with a set of tools to help make decisions about virtualization, taking into account performance requirements, costs, and business priorities.

The primary focus of this paper is the application of formal modeling techniques to reduce costs in IT middleware virtualization scenarios. The specific contributions of the paper can be stated as follows: (a) we identify the different factors impacting costs in a virtualization framework, and develop a formal mathematical cost model that takes into account both cost and quality of service under varying demand conditions; (b) we develop runtime models of three virtualization decision-making paradigms: (1) a *historical* model that assumes that future load patterns will closely follow past load trends, (2) a *staged* model that is able to handle unforeseen load patterns, and (3) a *mixed* model that attempts to combine the benefits of both the historical and staged models; (c) we describe the cost implications of each runtime model, based on the cost model, to help managers decide which decision-making approach to use; (d) we present a computational analysis of the impact of various managerial and situational factors on the overall cost of a virtualization infrastructure, and develop a set of insights to help IT managers to select appropriate resource and application parameters.

Our experiments demonstrated that the staged approach will result in about 20–30% higher costs than the near-optimal result. To combine the benefits of both the staged approach and the historical load based approach, we developed a mixed approach, and demonstrated that the mixed approach results in total cost within 7% of the near-optimal cost at moderate deviations from historical loads. In a set of sensitivity experiments, we show the impact of variation in allocation/deallocation costs, demand, and budget. Further, we include with our results a series of managerial insights that managers can use to decide which approach to use for resource allocations in a virtualized environment, including situations where the choice may vary based on stakeholder incentives.

The remainder of this article is organized as follows. Section 2 describes closely related work. Section 3 describes the problem, introduces our cost model. Section 4 describes our solution approach, and develops an efficient solution procedure based on a modification of dynamic slope scaling techniques. Section 5 discusses approaches to handle unforseen load scenarios. In Section 6, we describe some insights. In Section 7, we conclude the paper.

2. Related work

In this section, we describe related work in (a) scheduling and (b) grid computing, and contrast these approaches with our approach.

2.1. Scheduling

One pertinent research stream is scheduling, or the allocation of limited resources to optimize certain objective functions, such as on-time delivery of jobs (for a detailed treatment of the scheduling literature, see (Lee et al., 1997; Herroelen et al., 1998; Mokotoff, 2001; Cai and Zhou, 1999)). Recent developments in scheduling theory (Chen and Vairaktarakis, 2005) focus on more practical constraints, most of which are $\mathcal{NP}$-hard and demand the heuristic methods that have been the focus of attention for the past decade. Resource-constrained project scheduling refers to the scheduling of activities subject to precedence and resource constraints. The scheduling problem as related to production planning (Pochet and Vyve, 2004; Vollmann et al., 2004) also appears to be relevant.

There are, however, several major differences between the scheduling and planning research areas and the virtualization problem we are studying. We describe these below.

First, in most scheduling problems, each job has a well-defined start and end time, unlike in a virtualized application environment where applications run continuously under varying demand conditions. Requests arrive in a stream, and the processing time required to service a given request in the stream is not known a priori.

Second, production planning work considers labor and equipment as fixed costs. In IT environments, however, labor is often a direct cost for the purposes of chargeback frameworks (McKinnon and Kallman, 1987). For outsourced data centers, the hosting company assesses personnel-based charges based on the actual time required to service/maintain resources. Further, IT hardware and software resources become outdated quickly, and need to be refurbished or replaced every few years. In many cases, these resources are not purchased, but leased. Thus, unlike traditional manufacturing and production engineering, IT operational costs are not sunk costs.

Third, unlike many existing scheduling scenarios, the parties responsible for IT applications and the resources on which they will run may be different. This leads to a situation where the each application owner is responsible for the final cost of the applications they own, whereas it is the resource owners' responsibility to manage the cost of resources.

Finally, the heterogeneity of cost structure across various locations makes the middleware virtualization model complex. Unlike other resource scheduling problems, where resources reside together in a single location, a virtualized IT environment allows resources from multiple locations to be combined to achieve the desired demand of the application. This makes the modeling of our problem unique.

While our problem requires specialized attention, as we demonstrated above, there are several aspects of scheduling and planning research that can inform our middleware virtualization cost optimization problem.

In middleware virtualization, there is a cost to bind application to resources (we will define this cost formally in Section 3). In this respect, existing scheduling literature that considers setup costs, such as Havill and Mao (2008) and Allahverdi et al. (2008), is relevant. As demonstrated in Allahverdi et al. (2008), most of these problems are NP-hard and require custom heuristics, which informs our approach in this work.

In our middleware virtualization problem, quality of service (QoS) is a concern – for example, applications must provide response times within an agreed-to average response time. In Yao et al. (2008), the authors model resource allocation in wireless network by applying queuing theory. This is difficult to apply in an IT scenario due to the systems-oriented nature of the problem. In our model, we take a different approach by considering the application demand for various types of resources, and treating response time QoS as a constraint that must not be violated.

The online nature of the applications is a key aspect of the middleware virtualization resource allocation problem. In such a scenario, any solution approach must run quickly (i.e., on the order of seconds, not hours) so that it can run frequently to recognize and respond to changes in demand. Some recent scheduling work, such as Havill and Mao (2008), Ridouard et al. (2008), Averbakh (2010) and Kumar et al. (2006), considers online scenarios, and is therefore relevant. In these works, the authors propose heuristics to address online scheduling problems. In this work, we follow their lead, and develop heuristics for our own online problem.

Finally, some recent scheduling research (Polyakovsky and M'Hallah, 2009) takes an agent-based approach. Here, each agent is responsible for solving a version of the scheduling problem that maps to a portion of the whole problem. In the virtualization domain, an agent-based approach might map to each application owner making application binding decisions for owned applications. However, in industry, it is the data center manager who

makes allocation decisions based on the states of all applications and resources. Therefore, we take a centralized approach, where all decisions are made with the support of the application placement decision support module we propose in this paper, depicted in Fig. 2.

2.2. Grid computing and server virtualization

Grid computing and server virtualization problems are similar to middleware virtualization problems in that both problem areas look to make resource allocation decisions. In this section, we consider the similarities and differences between these areas in the context of our virtualization problem.

The authors of Santos et al. (2002) propose a scheme to assign resources to applications that is similar to our approach. However, these authors do not consider the business costs associated with running applications in a virtual data center, though these costs represent a key decision-making factor for most businesses. Rather, the authors assume that the network cost is the main component of application performance in a virtualized system and provide optimal allocation recommendations of applications to different parts of the network. However, given the glut of bandwidth (gigabit-sized ethernets are common in IT data centers) and declining prices of telecommunications infrastructures, network costs are not as significant as other costs. In contrast to this work, we focus on the dollar costs of maintaining and running applications in multiple data centers.

IBM, as part of its Alphaworks project, proposes a scheme called "Application Performance Evaluator and Resource Allocator" (IBM Inc., 2006a), which uses a mean value analysis and queuing theory approach based on performance metrics, such as CPU and disk utilization, as a basis for allocation decision-making. This approach, though innovative, does not consider important operational costs in the decision process as we do in our approach.

In Wolski et al. (2001), the authors investigate the economics of controlling resource allocation in a computational grid setting at a macro level. They consider the overall market of grid computing and study its viability and stability. Others (Bapna et al., 2008; Buyya et al., 2002) consider economic optimization models that ensure fairness among buyers and sellers of computing time.

Much research (Abramson et al., 2000; Berman and Wolski, 1997; Litzkow et al., 1998; Shao et al., 2000) in the grid computing literature postulates a *resource selector service* that can selects the application-to-resource binding on the basis of an application's characteristics. The service assists the mapping of the application workload to virtual machine resources and consists of three interrelated steps: selection, configuration, and mapping. Only after a mapping has been determined can the selector gauge whether one selection is better than another. In our case, the cost of changing resource-to-application bindings precludes us from considering approaches where mappings are evaluated only after the mapping has been tried.

Several authors (He et al., 2003; Liu et al., 2002; Min and Maheswaran, 2002) have presented frameworks that pertain to resource selection and scheduling, but none considers the problem from an optimization perspective where the goal is to minimize costs while meeting performance requirements. In Kumar et al. (2009), the authors describe scheduling computational jobs in grid computing from an economic stand-point. In this scenario, the jobs have a fixed start time and end time. In contrast, our research focuses on multiple resources required to run complex 24×7 enterprise applications. In our opinion, thus, the problem of resource allocation in middleware virtualization from the operational cost perspective has not been addressed in detail.

While the scheduling and grid/virtualization literature serves as a strong starting point for addressing our middleware virtualization problem, we cannot directly apply this work because the assumptions underlying traditional scheduling work do not carry over to the middleware context (as we describe above). In this work, we develop an optimization model and heuristic that, while based on existing work, will address the middleware context and incorporate the particular variables that impact it.

Most work in virtualization optimization and, indeed in scheduling in general, considers the technology in isolation of the organizational context, e.g., in our problem scenario, without considering how organizational factors can impact the realization of the economic benefits promised by virtualization. We present a discussion of some of these factors in Section 1, where we discuss how differing incentives across stakeholders and different operating models can confound the problem. While addressing all of these factors is outside the scope of our work here, we note that these and other factors represent a broad area of potential future research for the Operations Research community. We intend to pursue this line of inquiry in our future work.

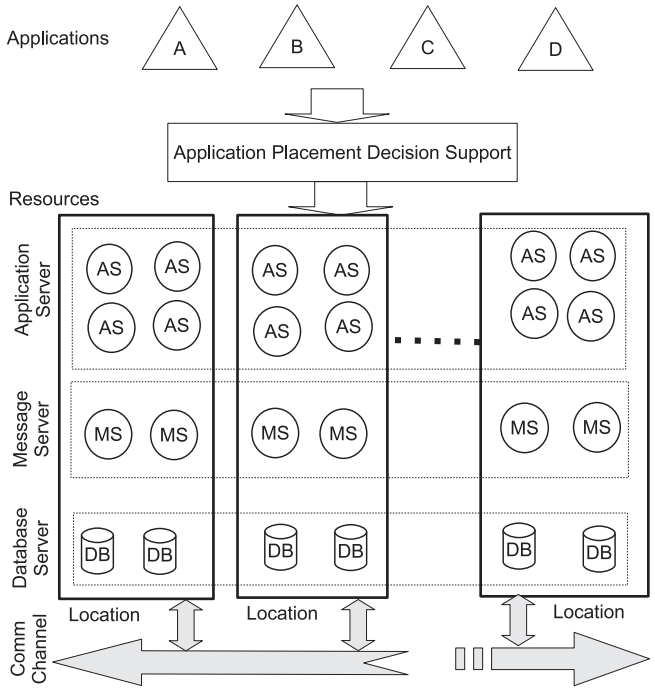

Fig. 2. Middleware virtualization architecture.

3. Problem model

A typical middleware virtualization environment consists of a set of resources servicing a set of applications, as depicted in Fig. 2. Multiple types of resources may be available in such an environment, such as Application Servers (AS), Message Servers (MS), Database Servers (DB), and others. These resources are distributed across various geographically separated data centers, as denoted by the different locations in Fig. 2. A networked communication channel connects these locations to facilitate communication between resources at different locations.

An Application Placement Decision Support (APDS) module monitors activity in each data canter to help the data center manager allocate resources to applications requiring those resources. The goal of this paper is to describe a scheme that the APDS module can use to provide data center managers with the information they need to allocate resources to minimize total cost of ownership.

To model the variation of the different costs over time effectively, we segment the duration under consideration (e.g., a day or half of a day) into several time periods, e.g., night, early morning, morning, noon, afternoon, and evening (time can be segmented as needed, depending on the variation of load in a particular data center setting). We assume that the costs remain fairly constant within a time period. This assumption is realistic because it represents widely acceptable and applied practice for various data centers and in the literature (Bapna et al., 2008; Ransbotham et al., 2010). We also assume that the time periods are of equal length (i.e., if there are six time periods in a 24-hour day, each time period represents 4 hours). We use T to denote the number of time periods and N to denote the duration of each period.

Our model's treatment of time, as described above, is generic, and can be adapted for demand variation based on seasonality, weekday/weekend loads, or other needs. Such an adaptation would require domain-specific knowledge of expected load trends for applications and data centers, which could be stored in a database or flat file and retrieved by the model to segment time as appropriate at runtime.

3.1. Resources

Resources are elements that are used by applications to run. Resources can be defined at various levels. Fig. 1 defines a typical resource stack. At the bottom of the resource layer is a set of hardware, such as memory, CPU, and storage. An operating system runs on the hardware to provide generic computing services. Within the operating system, various middleware software instances (e.g., database server, web server, application server) provide generic application support services. Applications are deployed to these middleware stacks at runtime. The application-to-resource binding of interest in this work happens at these top two layers, i.e., we are interested in binding applications to middleware stack resources.

In this paper, we focus on these middleware runtime environments and refer them as resources. In our discussion, the cost of these runtime environments includes the cost of underlying layers. For example, the cost of a database server will include the cost of the database server, the cost of the operating system where the database server is running, and the cost of the hardware where the operating system is running.

We consider only full allocation of a particular resource to an application; we do not consider any partial allocation. For example, if an application server is allocated to an application during a certain period, the application server is dedicated to that application for that period-no other application will share the same resources during the same period. Although the same resource can technically be shared by multiple applications at any time, there are certain risks associated with this, e.g., the low-level resource requirements (e.g., CPU usage) for one application may affect other applications running on the same resource at the same time. Therefore, large data centers typically isolate resources in terms of application service. Thus, for our model, resources are time-shared, but dedicated to at most one application at a given time (Rackspace Hosting, Inc., 2010).

3.1.1. Resource cost

Although some may argue that IT resources are sunk costs, recurring operating costs are often in excess of 80% of the IT budget, far outweighing capital spending (Masiero, 2006; Acey, 2006).

For our model, we are interested in the costs associated with the resources, the capacity of these resources, and communication costs associated with communication between resources supporting an application virtualized across multiple locations.

The cost of a resource typically consists of two parts: capital cost and operating cost. The capital cost includes the costs associated with leasing space, and leasing/interest payments. These costs occur on a per-machine basis, e.g., when placing equipment in a co-location facility, the cost is assessed on a per-server basis. We denote f_{ri} as the capital cost associated with the resource r at location i. The operational cost for a resource represents the cost of maintaining it. This cost includes the cost of power and the cost for the operational staff involved, which is typically charged to the application owner on a time and materials basis via a charge-back system (McKinnon and Kallman, 1987). This cost varies by geographical location.

Based on the above discussion, we denote the variable cost of a single unit of resource r at location i at time t as v_{ri}^t per unit time.

We assume that there are sufficient resources to handle all demand. The model can be extended to handle resource-constrained scenarios by incorporating a penalty constraint to ensure that high-priority applications meet demand.

3.1.2. Resource capacity

A limited number of resources are available at each location, and applications obviously cannot be allocated to more resources than are available. We define this conflict as the maximum capacity of a resource at a particular location. For instance, if a location has twenty middleware stack instances, its capacity is 20. We use C_{ri} to denote the capacity of resource r at location i, i.e., the maximum number of instances of the resource available at a location.

In real scenarios, some resources may be partially available at certain times, e.g., for maintenance. For the sake of our model, we assume that the total capacity of a resource at a particular location remains constant over time. The unavailability of a resource (or the variance of the total capacity of a resource at a particular location) can be modeled with a "ghost" application that requires that resource.

We assume that resources in the data centers are homogeneous, which enables the resources to be considered as a pool of resources, any one of which can be bound to an application. This binding decision is complex, because these data centers are geographically located at various locations, and the cost of resources in each of these data centers varies by location.

It is possible to consider heterogenous resources, where each resource type may have a different configuration (e.g., more RAM, or less storage space) in different resource instances. This would require that the model have access to the specific characteristics of each resource, and an additional set of constraints to ensure that an application's demand requirements do not exceed the capacity of the resources where it will run.

3.1.3. Communication

The data centers are connected with a communication network. The network is used to communicate across applications deployed in multiple locations/data-centers. Typically the network is leased line (like a T1 or T3 line), the cost of which is fixed over a month or a year. Therefore, we do not include any representation of communication costs in our model. We further assume that the communication links between data centers have sufficient bandwidth to handle traffic to and from the data center. We could extend our model to handle constrained bandwidth with an additional constraint to ensure that the sum of expected data traffic for all applications running in a data center does not exceed the capacity of the channel.

3.2. Applications

Applications exhibit various demand characteristics over time that affect resource usage for the application. When demand

decreases, some resources may be freed and reallocated to other applications. However, when a resource is deallocated from one application and allocated to another, a cost ensues. Typically, this two-part de-and re-allocation cost involves human and/or tool intervention to set up the new application-to-resource binding.

3.2.1. Cost of allocation/deallocation of applications to/from resources

The cost of deallocating a resource from an application refers to the costs associated with unbinding the application from the resource. Similarly, the cost of allocating a resource to an application refers to the costs associated with binding the application to the resource.

Both allocation and deallocation costs consist of fixed and variable components. Consider deallocation as an example. If an application is using 100 application server instances, and the new demand level requires only 50 instances, 50 instances can be freed. Although various software tools provide assistance, human effort is needed to interact with the software to actually deploy/undeploy the application, and verify the outcome of the stop/start action. The deallocation cost is the cost of the tool and human effort involved in freeing 50 application server instances from the application binding. A fixed amount of work must occur, regardless of how many instances are deallocated, e.g., in undeploying the application from the 50 application server instances. The variable cost depends on how many instances are being deallocated, e.g., checking each application server instance to ensure proper behavior after the deallocation.

Based on this discussion, the deallocation cost of x units of resource r from application j can be given as $\alpha_{rj} + \beta_{rj}x$, where, α_{rj} is the fixed and $\beta_{rj}x$ is the variable part of the deallocation cost. Similar to the variable operating cost, because personnel costs may vary depending on the time and location at which the deallocation occurs, we generalize the cost of deallocating x units of resource r from an application j at time t and location i to be $\alpha^t_{rij} + \beta^t_{rij}x$.

The formulation for the allocation cost parameter is similar to that of the deallocation cost. For example, allocating additional application server instances to an application consists of a fixed component (deploying the application to the new resources) and a variable component (checking the newly allocated instances for proper behavior after deployment). Considering both the fixed and variable costs, the allocation cost of x units of resource r to application j at time t and location i is given as $\gamma^t_{rij} + \delta^t_{rij}x$, where γ^t_{rij} is the fixed portion of the allocation cost and δ^t_{rij} is the variable part of the allocation cost. Note that allocation and deallocation costs are expressed on a per-event basis in the model.

3.2.2. Application demand

Each application places a specific level of demand on different types of resources, where demand represents the number of resources required to meet Quality of Service (QoS) levels, e.g., as specified in a service level agreement. For example, if an application requires ten application servers and two database instances in order to meet response-time service level requirements, the demand for application server resources is 10 units, and the demand for database server resources is 2 units.

We denote d^t_{rj} as the demand of application j for resource r during time period t. Because a given unit of a resource is allocated to a single application at a given time, the amount of resource that can be allocated or freed is an integer. Thus, our model requires a constraint to ensure that the demand for a resource type across all applications at a particular location is less than or equal to the maximum capacity of that resource at that location.

3.2.3. Budget of the application

In many enterprise scenarios, an application is owned by a specific department, which is responsible for bearing the cost of the application. These costs are allocated to an application through chargeback schemes (McKinnon and Kallman, 1987; Vanover, 2007; Pearlson and Saunders, 2006). A budget (B_j) is allocated for each individual application (j) by the application owner, and the total cost of running the application must remain within that budget.

The application owners are responsible for paying the usage cost of the resource, which in our model includes: (a) the operational cost of usage across all resources and (b) the allocation and deallocation cost for each such event. The capital cost of the resource is incurred by the resource owners irrespective of applications' use of the resource. If a resource is not being utilized at all, it can be taken out of service, thus saving the fixed dollar costs for the space, lease payment, etc.

3.3. Decision problem at application placement controller

The decision problem for the APDS module can be stated as follows: *Given a set of resources located at a set of locations and a set of applications with a demand for resources and set budget levels, determine the resource-to-application binding in such a way that the total cost of allocation of the various resources to applications is minimized while demand and budget constraints from all applications are also satisfied at all times.*

We assume that the set of applications changes infrequently. For example, in the credit card company's data centers, applications are added, updated or removed relatively rarely (e.g., approximately once a month), and only during very low usage periods, e.g., very early in the morning.

3.4. Cost optimization model

In this section, we concentrate on the business goal of minimizing the dollar cost of running a set of applications in a virtual IT environment with specific QoS expectations irrespective of demand workload, budget constraints, and resource capacity constraints. Before delving into the details, we first provide a list of the notation we use throughout the rest of the article.

Indices

r	index for resources $r = 1,\ldots,R$
i	index for locations $i = 1,\ldots,I$
j	index for applications $j = 1,\ldots,J$
t	index for time periods $t = 1,\ldots,T$

Parameters

d^t_{rj}	units of demand for resource r from application j in period t
f_{ri}	capital cost for using resource r at location i for the duration of the period (cents)
v^t_{ri}	operating cost of using resource r at location i in period t (cents/second)
α^t_{rij}	fixed cost for deallocating resource r at location i from application j in period t (cents)
β^t_{rij}	variable cost for deallocating one unit of resource r at location i from application j in period t (cents)
γ^t_{rij}	fixed cost for allocating capacity resource r at location i from application j in period t (cents)
δ^t_{rij}	variable cost for allocating one unit of resource r at location i from application j in period t (cents)
C_{ri}	number of units of resource r at location i
T	number of time periods
N	duration of time period
B_j	budget of application j for the duration of the period (cents)
x_{rij}	initial units of resource r at location i that is allocated to application j

Variables

F^t_{rij}	capacity of resource r at location i deallocated from application j at the end of period t
A^t_{rij}	capacity of resource r at location i allocated to application j at the beginning of period t
W^t_{rij}	=1 if $F^t_{ij} > 0$, and 0 otherwise
V^t_{rij}	=1 if $A^t_{ij} > 0$, and 0 otherwise
Y_{ri}	=1 if resource r at location i is ever used, and 0 otherwise

3.4.1. Minimum dollar cost model

Problem **P** models the cost optimization as an integer program.

Problem P

$$\mathbf{Z}(\mathbf{P}) = \min \sum_{ri} f_{ri} Y_{ri} + \sum_{rijt} v^t_{ri} N \left(\sum_{\tau=1}^{t} A^\tau_{rij} - \sum_{\tau=1}^{t-1} F^\tau_{rij} + x_{rij} \right) + \sum_{rijt} \left(\alpha^t_{rij} W^t_{rij} + \beta^t_{rij} F^t_{rij} + \gamma^t_{rij} V^t_{rij} + \delta^t_{rij} A^t_{rij} \right) \quad (1)$$

subject to

$$\sum_{j} \left(x_{rij} + \sum_{\tau=1}^{t} A^\tau_{rij} - \sum_{\tau=1}^{t-1} F^\tau_{rij} \right) \leqslant C_{ri}, \quad \forall r,i,t, \quad (2)$$

$$x_{rij} + \sum_{\tau=1}^{t} A^\tau_{rij} \geqslant \sum_{\tau=1}^{t} F^\tau_{rij}, \quad \forall r,i,j,t, \quad (3)$$

$$\sum_{i} \left(x_{rij} + \sum_{\tau=1}^{t} A^\tau_{rij} - \sum_{\tau=1}^{t-1} F^\tau_{rij} \right) \geqslant d^t_{rj}, \quad \forall r,j,t, \quad (4)$$

$$\left(x_{rij} + \sum_{\tau=1}^{t} A^\tau_{rij} - \sum_{\tau=1}^{t-1} F^\tau_{rij} \right) \leqslant M^1_{rij} Y_{ri}, \quad \forall r,i,j,t, \quad (5)$$

$$F^t_{rij} \leqslant M^2_{rijt} W^t_{rij}, \quad \forall r,i,j,t, \quad (6)$$

$$A^t_{rij} \leqslant M^3_{rijt} V^t_{rij}, \quad \forall r,i,j,t, \quad (7)$$

$$x_{rij} + \sum_{\tau=1}^{t} A^\tau_{rij} - \sum_{\tau=1}^{t-1} F^\tau_{rij} \leqslant M^4_{rijt} Z^t_{rij}, \quad \forall r,i,j,t, \quad (8)$$

$$\sum_{rit} v^t_{ri} N \left(\sum_{\tau=1}^{t} A^\tau_{rij} - \sum_{\tau=1}^{t-1} F^\tau_{rij} + x_{rij} \right) + \sum_{rit} \left(\alpha^t_{rij} W^t_{rij} + \beta^t_{rij} F^t_{rij} + \gamma^t_{rij} V^t_{rij} + \delta^t_{rij} A^t_{rij} \right) \leqslant B_j, \quad \forall j, \quad (9)$$

$$A^t_{rij}, F^t_{rij} \geqslant 0 \ \& \ \texttt{integers}, \quad \forall r,i,j,t, \quad (10)$$

$$Y_{ri}, W^t_{rij}, V^t_{rij} \in \{0,1\}, \quad \forall r,i,j,t, \ \forall k \in I. \quad (11)$$

$M^1_{rij}, M^2_{rijt}, M^3_{rijt}$ and M^4_{rijt} are big-M coefficients to create relationships between integer variables and corresponding binary variables.

The total dollar cost of running an application consists of (a) the capital cost of using a resource, (b) the operational cost of using a resource for an application, and (c) the costs of deallocating or allocating a resource. The first term in the objective function is the capital cost associated with the use of resource r at location i, which depends only on whether the resource at that location is used. The second term is the operational cost associated with the use of a particular resource at a location by an application, which depends on the use of the resource by each application allocated to it. The third term in the objective function captures the variable and fixed cost of allocation and deallocating resources to/from applications.

Constraint (2) ensures that resource allocations never exceed the available capacity for any time period. Constraint (3) ensures that the amount deallocated from a resource is less than the amount allocated to a resource. Constraint (4) ensures that the demand of an application for a particular resource is always met. Constraint (5) ensures that if a resource is allocated to an application, the fixed cost for that resource (f_{ri}) is assessed. Constraint set (6) and (7) guarantees that if a resource is allocated or deallocated, the appropriate variables to account for the fixed allocation–deallocation costs are triggered. Constraint (9) restricts the cost of an application owner to the budget of the application. Constraint set (10) and (11) specifies the range of values for the decision variables.

Finally, we note that it is possible to further tighten the big-M coefficients; however, this is straightforward, and is left as an exercise for the reader.

4. Solution approach

In this section, we consider how we might enable the APDS module to use the model to support a data center manager's decision-making process. We first consider the complexity of our model (discussed in Section 3) and show that it maps to the fixed-charge transportation problem, an $\mathcal{NP}$-hard problem, making a direct application of the model intractable. We then consider recent work in developing heuristics for similar problems, and show why these cannot be directly applied. Finally, we develop a heuristic for our problem based on the Dynamic Slope Scaling Procedure and show that our heuristic produces an allocation that is close to optimal, with running times fast enough to support interactive decision support.

We first show that the allocation problem described above is a $\mathcal{NP}$-hard problem.

Theorem 1. *The minimum cost problem* **P** *is* $\mathcal{NP}$*-hard.*

Proof. The proof is based on a reduction of the $\mathcal{NP}$-hard fixed-charge transportation problem,

$$\begin{aligned} \min \quad & \sum_i \sum_j c_{ij} x_{ij} + f_{ij} y_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} \leqslant S_i \qquad \forall i, \\ & \sum_i x_{ij} \geqslant D_j \qquad \forall j, \\ & 0 \leqslant x_{ij} \leqslant M y_{ij} \qquad \forall i,j, \\ & y_{ij} \in \{0,1\} \qquad \forall i,j, \end{aligned}$$

where $i \in \{1,\ldots,I\}$ are indices for supply locations, $j \in \{1,\ldots,J\}$ are indices for demand locations, S_i is the supply at location i, D_j is the demand at location j, c_{ij} is the per-unit transportation cost from i to j, f_{ij} is the fixed transportation cost from i to j, and M is a big number. Given any case of this fixed-charge transportation problem, we can construct an instance of problem **P** with the following restrictions:

- There is single period $T = 1$;
- There is a single resource $R = 1$;
- The cost parameters $f_{1i} = v^1_{1i} = \alpha^1_{1ij} = \beta^1_{1ij} = 0$ for all i,j;
- The cost parameters $\gamma^1_{1ij} = f_{ij}$ and $\delta^1_{1ij} = c_{ij}$ for all i,j;
- The capacity $C_{1i} = S_i$, and the demand $d^1_{1j} = D_j$ for all i,j;
- The initial allocation $x_{1ij} = 0$ and,
- $B_j = 0$, $\forall j$.

In this case, the variables Y_{ri}, F^t_{rij}, and W^t_{rij} and the constraints 3, 6, 7, 9 can be eliminated; the resulting instance of **P** is equivalent to the fixed-charge transportation instance. The construction is polynomial, and the result then follows. □

Corollary 1.1. *The minimum cost problem* **P** *is* $\mathcal{NP}$*-hard, even under the following conditions: a single time period; one resource, infinite budget for applications; the fixed cost of resource usage is zero; the*

application incurs only allocation cost, no de-allocation costs. The corollary follows from the details of the proof.

Even though we have shown above that our problem is $\mathcal{NP}$-hard, we attempted to solve it for small-to-medium problem sizes using CPLEX 12.0 (ILOG, 2009) on a 2.2 GHz dual processor Windows XP machine with 4 GB RAM. We found that it took on the order of 30–60 minutes to obtain an optimal solution for a single instance of a small problem size (number of locations = 5, number of applications = 30, number of time periods = 4 and number of resources = 4) in this environment. These running times are far too long for an interactive decision support environment. We tried problem sizes closer to expected real-world problem sizes (number of locations = 10, number of applications = 100, number of time periods = 6 and number of resources = 6), but could not obtain optimal solutions (we ran out of memory in the test environment). Therefore, we consider alternate approaches and heuristics to solve the problem.

The virtualization problem can be considered to be a multi-stage fixed-charge network flow problem. Each stage of this network flow denotes a time-period. In each time-period we have several applications, represented by nodes. An arc from node i to another node j denotes a resource is being deallocated from node i and is being allocated to node j. There is a cost associated with each arc that includes the fixed cost and variable cost of allocation and deallocation as described before. The two popular approaches to solving the fixed charge network flow problem are the Hybrid approach proposed by Kim and Hooker (2002) and the Dynamic Slope Scaling Procedure (DSSP) proposed by Kim and Pardalos (1999).

The Hybrid approach (Kim and Hooker, 2002) takes about 32 hour to solve a problem size of 18 origin and 18 destination nodes, which is very small compared to the problem size we are addressing. A 100-application virtualization problem would have 100 origins and 100 destinations in a single time period. With multiple time periods and resources, the virtualization problem becomes so complex that the Hybrid approach proposed in Kim and Hooker (2002) cannot solve it, even in a few hours. This, too, is too slow for interactive decision support.

While we can apply the broad DSSP approach to solve problem **P**, some of the problem's characteristics require modification to the base DSSP approach to be able to develop solutions.

The original DSSP is motivated by an economic viewpoint (marginal concept) of the fixed cost. It approximates a solution for the fixed charge network flow problem by solving successive LP problems with recursively updated objective functions, where fixed costs are replaced by a variable cost. In each iteration, the fixed cost is updated based on the LP result that effectively reflects the current marginal variable cost and the fixed cost of the solution. These LP problems can be solved efficiently since the set of constraints is not changed and all the binary variables from the mixed integer formulation are removed. In this approach, solving a fixed-charge problem can be interpreted as finding a break-even point (level of activities) to justify the investment of fixed costs. Iterations proceed until there is no further improvement, or until the number of iterations reaches a preset maximum. At this point, the linear approximation costs of the final solution correspond to the true objective function.

Algorithm 1 is a modified version of DSSP designed to suit problem **P**. We describe the three main modifications to the base DSSP logic below.

First, unlike many other multicommodity problems, the variables associated with variable cost, A and F, are integer variables. Demand is also an integer variable. Therefore, we cannot use an intermediate LP formulation in our modified DSSP approach to solve the problem **P** (as is usually the case when applying DSSP). Rather, we need an IP formulation – mDSSP (**P**), given below in Expression (12).

Second, typical applications of the DSSP approach do not include cost functions within the constraint set. Expression (12) in the IP mDSSP (**P**) is a modification to the budget constraint given in Expression (9) to appropriately represent fixed costs. We solve mDSSP (**P**) by iteratively applying relaxation and integer approximation to generate a near-optimal solution of DSSP (**P**). This appears in Algorithm 1 in lines 8–19 (in a typical DSSP approach, this would be a simple LP).

Problem mDSSP(P)

$$\begin{aligned}\mathbf{Z}(\mathbf{mDSSP}(\mathbf{P})) = \min \quad & \sum_{rijt} f'_{ri} A^t_{rij} \\ & + \sum_{rijt} \left(v^t_{ri} N \left(x_{rij} + \sum_{\tau=1}^{t} A^{\tau}_{rij} - \sum_{\tau=1}^{t-1} F^{\tau}_{rij} \right) \right) \\ & + \sum_{rijt} \left(\left(\alpha'^t_{rij} + \beta^t_{rij} \right) F^t_{rij} + \left(\gamma'^t_{rij} + \delta^t_{rij} \right) A^t_{rij} \right) \end{aligned} \tag{12}$$

subject to

$$\sum_{rit} v^t_{ri} N \left(\sum_{\tau=1}^{t} A^{\tau}_{rij} - \sum_{\tau=1}^{t-1} F^{\tau}_{rij} + x_{rij} \right) + \sum_{rit} \left((\alpha'^t_{rij} + \beta^t_{rij}) F^t_{rij} + (\gamma'^t_{rij} + \delta^t_{rij}) A^t_{rij} \right) \leqslant B_j, \quad \forall j \tag{13}$$

and (2)–(5), (8), (10), (11).

Third, the solution approach using an IP formulation includes the possibility that intermediate problems may become infeasible. Algorithm 1 includes verification logic (lines 21 and 39) to check for this condition and ensure that processing proceeds.

Algorithm 1. (Modified DSSP).

1: Initialize fixed costs
$f'_{ri} = \frac{f_{ri}}{\sum_{jt} \min\{C^0_{ri}, \Delta^t_{rij}\}}, \ \forall r,i; \ \alpha'^t_{rij} = \frac{\alpha^t_{rij}}{\sum_{jt} \min\{C^0_{ri}, \nabla^t_{rij}\}}, \ \forall r,i,j;$
$\gamma'^t_{rij} = \frac{\gamma^t_{rij}}{\sum_{jt} \min\{C^0_{ri}, \Delta^t_{rij}\}}, \ \forall r,i,j$
2: `Set` *DSSP_Iteration_Counter* := *max_dssp_iterations*
3: `Set` *UpperBound* := *+INFINITY*
4: `Set` *LowerBound* := *−INFINITY*
5: **while** *DSSP_Iteration_Counter* > 0 **do**
6: `Set` *DSSP_Iteration_Counter* := *DSSP_Iteration_Ccounter* − 1
7: `Set` *IP_Iteration_Counter* := *max_ip_iterations*
8: **while** *IP_Iteration_Counter* > 0 **do**
9: `Set` *IP_Iteration_Counter* := *IP_Iteration_Counter* − 1
10: `Solve integer relaxation solution of problem` **mDSSP(P)**
11: **if** `Integer relax solution available` **then**
12: **for** ∀*r,i,j,t* **do**
13: **if** $\left(ceil\left(A^t_{rij}\right) - A^t_{rij} < 0.15\right)$ `Fix` $A^t_{rij} := ceil\left(A^t_{rij}\right)$
14: **if** $\left(ceil\left(F^t_{rij}\right) - F^t_{rij} < 0.15\right)$ `Fix` $F^t_{rij} := ceil\left(F^t_{rij}\right)$
15: **if** $\left(A^t_{rij} - floor\left(A^t_{rij}\right) < 0.15\right)$ `Fix` $A^t_{rij} := floor(A^t_{rij})$
16: **if** $\left(F^t_{rij} - floor\left(F^t_{rij}\right) < 0.15\right)$ `Fix` $F^t_{rij} := floor(F^t_{rij})$
17: **end for**
18: **end if**
19: **end while**
20: `Solve the problem` **mDSSP(P)**
21: **if** `integer solution available` **then**

(*continued on next page*)

Algorithm 1. (Modified DSSP).

22: *currentUB* `is set to the solved value of` **mDSSP(P)**
23: **if** *currentUB* < *UpperBound* **then**
24: `Set` *UpperBound* := *currentUB*
25: `Mark the current solution as the Upper-Bound solution of the problem` **DSSP(P)**
26: **end if**
27: **for** $\forall r,i,j,t$ **do**
28: **if** $\left(A^t_{rij} > 0\right)$ `Fix` $V^t_{rij} := 1$ **else** `Fix` $V^t_{rij} := 0$
29: **if** $\left(F^t_{rij} > 0\right)$ `Fix` $W^t_{rij} := 1$ **else** `Fix` $W^t_{rij} := 0$
30: **end for**
31: **for** $\forall r,i$ **do**
32: **if** $\sum_j x_{rij} + \sum_{tj} A^t_{rij} + \sum_{tj} F^t_{rij} > 0$ **then**
33: `Fix` $Y_{ri} := 1$
34: **else**
35: `Fix` $Y_{ri} := 0$
36: **end if**
37: **end for**
38: `Solve integer relaxation of the problem` **P**
39: **if** `LP solution available for problem` **P then**
40: *currentLB* `is set to the solved value of the LP relaxation of` **P**.
41: **if** *currentLB* > *LowerBound* **then**
42: `Set` *LowerBound* := *currentLB*
43: **end if**
44: **end if**
45: **end if**
46: `Update the fixed cost values`

$$f'_{ri} = \begin{cases} \frac{f_{ri}}{m_{jt}\widetilde{A}^t_{rij}}, & \text{if } \sum_{jt} \widetilde{A}^t_{rij} > 0 \\ 0, & \text{otherwise} \end{cases} \quad \forall r,i,j,t$$

$$\alpha'^t_{rij} = \begin{cases} \frac{\alpha^t_{rij}}{\widetilde{F}^t_{rij}}, & \text{if } \widetilde{F}^t_{rij} > 0 \\ 0, & \text{otherwise} \end{cases} \quad \forall r,i,j,t$$

$$\gamma'^t_{rij} = \begin{cases} \frac{\gamma^t_{rij}}{\widetilde{A}^t_{rij}}, & \text{if } \widetilde{A}^t_{rij} > 0 \\ 0, & \text{otherwise.} \end{cases} \quad \forall r,i$$

47: **end while**

To test the effectiveness of our solution procedures, we ran our solution method with various parameter values. The parameter values were chosen on the basis of data collected while working with the credit card company and other industries (a major communication company, a major online retail company and a virtualization company). These parameter values were also justified by data provided by various reports on virtualization implementations and case studies (Intel Inc., 2008; Aberdeen Group, 2008).

In the situation of the credit card company, each application uses several available resources, including database servers (DB2), remote web services, security services, LDAP services, transactional services (WebSphere MQ), application servers (WebSphere), and storage systems. Instances of these resources are located in multiple data centers in four locations (New Jersey, Dallas, San Jose, and Atlanta). Each location has two physically separate data centers, for a total of eight data centers (locations in our model). The number of applications the data centers run at any particular point in time varies between 90 and 120. The application demand variation over the course of a day can be segmented into multiple time slots. This architecture is typical for large enterprises.

To demonstrate the efficacy of the modified DSSP, we first ran a set of experiments to demonstrate the effectiveness of our solution procedure for small problems, where it is possible to obtain an optimal solution, comparing the cost obtained by the *Modified-DSSP* approach with that obtained by the optimal IP solution (using CPLEX). We found that the average gap between the modified DSSP case and the optimal IP case is below 0.3%. The average gap between modified DSSP case and the lower bound is below 4%. This demonstrates that our modified DSSP approach generates a near-optimal solution to the minimum cost problem.

Based on these results, we now show the effectiveness of our solution procedure for large problems for which it was not possible to obtain an optimal solution. We select problem sizes based on realistic infrastructures, using two values for number of locations (8,10), two values for number of applications (80,100), two values for number of resources (6,8) and two values for number of periods (6,8), for a total of 16 cases. For each case, we generate 10 problem instances by randomly selecting various parameters within ranges determined by the data collected from the data center of the credit card company. The ranges are $C_{ri} \in [50,100]$, $v^t_{ri} \in [3,8]$, $f_{ri} \in [9,24]$, $\beta^t_{rij} \in [3,8]$, $\delta^t_{rij} \in [3,8]$, $\alpha^t_{rij} \in [9,24]$, $\gamma^t_{rij} \in [9,24]$, and $B_j \in [8000,9000]$.

Table 1
Model accuracy results.

# Locations	# Applications	# Resources	# Period	mDSSP Large size problems		Staged approach	
				Avg % LP gap	Time (second)	Avg % LP gap	Time (second)
8	80	6	6	2.9	28	24	2
8	80	6	8	2.4	29	25	2
8	80	8	6	2.9	27	23	2
8	80	8	8	2.8	28	24	2
8	100	6	6	2.7	29	25	2
8	100	6	8	3.1	29	27	2
8	100	8	6	3.2	29	27	2
8	100	8	8	2.9	27	26	2
10	80	6	6	3.3	30	27	3
10	80	6	8	3.2	30	28	4
10	80	8	6	3.1	31	28	3
10	80	8	8	2.8	30	28	3
10	100	6	6	3.1	38	29	4
10	100	6	8	3.4	38	30	4
10	100	8	6	3.5	37	30	4
10	100	8	8	3.3	38	29	4

For such problem sizes, it is not feasible to obtain IP solutions using CPLEX, so we compare the solutions from our modified DSSP approach to that of the lower bound solution obtained by DSSP (line 42 of Modified DSSP Algorithm), and present minimum, maximum, and average gaps between the solutions from these two approaches. Table 1 presents these results, along with the running time for the *Modified DSSP* algorithm on a Pentium 2.1 GHz processor.

The average gap lies below 3.1%, with a maximum gap of 4.4% across all cases. Based on results from small problem sizes, we anticipate that the larger gap percentage is mostly due to the larger gap between the lower bound and IP solutions, i.e., we expect that the modified DSSP algorithm produces near-optimal results for larger problem sizes. The average running time for the modified DSSP algorithm is around 35 seconds for all problem instances. This enables the *Modified DSSP* algorithm to be run more frequently. For example, if the variation of load for an application is high, it is possible to have a larger number of stages of shorter duration.

5. Handling variation in load

The solution approach presented in Sections 3 and 4, like all scheduling approaches of this type, assumes good knowledge of expected workloads. Thus, its effectiveness depends on the accuracy with which the actual load matches historical load trends. While application loads typically follow historical load trends in general, unforseen peaks and shallows do occur. In this section, we describe methods for handling such variation. We first present two modifications of our model that can be applied periodically, as needed, to determine resource allocations and run a set of experiments to determine the cost implications of using this approach. Second, we describe how this modification can be used along with the original model approach for improved results, shown experimentally. Finally, we develop a set of managerial insights based these experiments.

5.1. Staged approach

Here, we take a staged approach. Instead of solving the model for a long future duration based on predicted variation across multiple periods, we solve the model for a very short duration y, where the value of y is selected in such a way that the demand can be expected to remain stable during the period. At the beginning of the y-length period, we solve the model $Z(P)$ with the number of time periods $T = 1$, and the duration of a time period $N = y$. Because we do not know whether the demand will rise or fall at the end of the y-length period, we consider either allocation or deallocation at the beginning of period y. If the demand rises at the beginning of y (compared with the previous period), we consider the cost of allocating resources; if the demand falls at the beginning of y, we consider the cost of deallocating resources at the beginning of period y. After each y-length period, the model is again solved for next y-length period, and so on.

To judge the accuracy of the above approach, we compare the solutions generated by the staged algorithm with the solutions obtained by the *Modified DSSP* algorithm. We generate several problem instances by randomly selecting various parameters within ranges determined by the data collected from the data center of the credit card company. The ranges are $C_{ri} \in [50,100]$, $\nu^t_{rij} \in [3,8]$, $f_{ri} \in [9,24]$, $\beta^t_{rij} \in [3,8]$, $\delta^t_{rij} \in [3,8]$, $\alpha^t_{rij} \in [9,24]$, $\gamma^t_{rij} \in [9,24]$, and $B_j \in [8000,9000]$. In Table 1, we report the gaps between the solutions obtained by staged approach and the *Modified DSSP* algorithm, as well as the per-stage running time required for the staged approach.

The result obtained by staged algorithm is within 20–30% of the solution obtained by *Modified DSSP* algorithm. In terms of running time, each run of the staged algorithm takes 2–4 seconds, compared to 35 seconds for the *Modified DSSP* algorithm for all stages.

Insight 1: In a pure staged approach, IT managers can expect to incur 20–30% higher costs, as compared to near optimal costs in the original model. These costs represent the costs associated with not knowing the future load, and the associated some incorrect allocation/deallocation decisions.

5.2. Mixed approach

In the mixed approach, we superimpose the staged approach on the *Modified DSSP* approach to balance the near optimality available using historical load trends and the need to handle unforseen demand. In this approach, the system derives an initial solution based on historical load. While running, if the actual load of a subset of the full application set deviates from the historical load, we solve the problem $Z(P)$ for that subset of applications only. In modeling $Z(P)$ for that subset of applications, we follow the staged approach described in Section 5.1. The value of $N = y$ is chosen based on expected duration of the peak or shallow. In this case, problem $Z(P)$ needs to be solved only for those applications whose load varies from predicted levels and number of time periods $T = 1$, which results in a reduction of the solution generation time for $Z(P)$ to single-digit seconds. When the load of the application returns to the historical load, the predetermined allocation is followed.

Such an approach has two main advantages. First, since application demand tends to follow historical trends in general, it is unlikely that all applications will deviate greatly at the same time. Based on this, the runtime instance of $Z(P)$ is much smaller than the instance of $Z(P)$ that needs to be solved to determine the allocation scheme for the entire application set for several time periods. This results in lower running times. Second, the dynamic part of the approach focuses only on the application(s) where the actual demand varies from the predicted value, rather than on the full set of applications, thus accommodating any variation in actual load values from the historical trend with minimal overall deviation from the pre-determined allocation scheme.

To demonstrate the accuracy of the mixed approach, we generate several instances of the model. Each instance has six time periods, number of locations = 10, number of applications = 100 and number of resources = 8. For each combination of number of locations and number of applications (25 combinations), we generate three instances, for a total of 75 problem instances.

First, we solve the base model to pre-determine the allocation scheme based on expected demand values based on historical trends. Next, at runtime we introduce variations in actual load for a percentage of applications, and apply the staged approach and mixed approach to determine the actual cost incurred by the system due to these two approaches, respectively.

We introduce these load variations using two parameters *theta* and *Theta*. We use *theta* to introduce small variations in load to model stable, but not constant, load over a period of time. We use *Theta* to introduce major changes in load that occur at the start of a new period of stability. Every y seconds, we vary the load of each application by ±*theta* or ±*Theta* (with appropriate safeguards in place to prevent negative workloads). The frequency of selecting *Theta*-sized load changes models the variability of load against historical trends. As a baseline, we start with *Theta* changes that match expected loads based on historical trends. We then vary the frequency of introducing *Theta* changes by selecting a change period z, where z is the number of seconds until the next *Theta* change. We set z based on a variability parameter w, where large values of w indicate high variability (frequent load changes), and small values of w indicate more stable workloads, i.e., less frequent load changes. We choose z from a normal distribution with a mean of N/w.

By tracking the variations that occur in each problem instance while running, we can use the actual load values to develop a near-optimal solution for comparison purposes (using the actual loads as the historically-expected loads in the model). This allows us to model and compute the near optimal cost for the actual load scenarios after the fact, for comparison purposes, despite the fact that the reliance on historical loads becomes untenable when loads vary widely.

We compute the percentage gap for the staged approach and the mixed approach with respect to the near-optimal solution from the model. In Table 2, we report these two gaps (averaged), along with the percentage of applications for which the actual load varies from the historical load, and the average deviation of the actual load from the historical load for these applications.

The percentage gap of the staged approach does not vary with the percentage application for which actual load differs from the predicted load, or the average deviation of the actual load from historical load. This gap lies between 20% and 30%, the same range seen in Table 1. Clearly, the amount of deviation from the historically predicted load value does not impact the staged approach.

The gap between the solution obtained by the mixed approach and the near-optimal solution increases with both the percentage of applications that deviate and the average deviation. As the percentage of applications that deviate from predicted load increases, the gap increases. Similarly, as average load values deviate further and further from predicted values, the gap increases. The maximum gap of about 17% occurs when 50% of applications deviate from predicted load by an average of 50% from the predicted load value. In moderate cases, e.g., when 30% of applications deviate from predicted values, with an average deviation of about 30%, the gap is just 7%. When the actual load of only 10% of applications vary from the predicted load with average deviation of just 10%, the gap between the mixed approach and the near optimal approach from the model is 2%.

Insight 2: At moderate and low variations of the actual load from the predicted historical load, the mixed approach will incur about 6% higher costs than the near optimal cost found by the model on average. At higher variations, the advantage of using the mixed approach decreases because the effort of taking historical trends into account is wasted, and associated cost implications approach those of the staged approach. For dynamic applications, there is no point in comparing the actual load with the historical trend. Here, the staged approach can be used, incurring 25–30% higher costs, as compared to the near optimal cost. IT managers can use Table 2 as a reference to decide which scheme to use for resource allocation decision.

Table 2
Accuracy for the mixed approach.

Percentage applications	Average deviation	Avg % gap of staged approach	Avg % gap of mixed approach
10	10	23	2
10	20	22	3
10	30	27	3
10	40	22	5
10	50	22	7
20	10	25	2
20	20	26	4
20	30	28	5
20	40	29	9
20	50	28	9
30	10	27	2
30	20	21	6
30	30	22	7
30	40	30	10
30	50	29	12
40	10	21	3
40	20	25	8
40	30	22	9
40	40	21	11
40	50	22	13
50	10	27	4
50	20	21	7
50	30	22	12
50	40	27	12
50	50	28	17

6. Effect of variation in parameters

In this section, we describe the effect of various parameters for resources and applications in the system in order to develop a set of managerial insights to help managers understand the cost implications of virtualization technology.

To draw some meaningful insights, we look at the cost implications from the perspective of both the data center manager and the application owner. We try to understand the implications on the total cost value if parameter values controlled by the data center manager and the application owner are modified. Allocation–deallocation cost, operational and capital expenditure costs are controlled by data center manager, while budget and demand (and the effect of QoS constraints) for various resources for an application is controlled by the application owner. Though we ran a variety of experiments, we report only those that show interesting results.

For all experiments, we ran the modified DSSP using a baseline set of parameter values: $C_{ri} \in [500–800]$, $\nu^t_{ri} \in [6—11]$, $f_{ri} \in [30—55]$, $\beta^t_{rij} \in [6—11]$, $\delta^t_{rij} \in [6—11]$, $\alpha^t_{rij} \in [30—55]$, $\gamma^t_{rij} \in [30—55]$, $B_j \in$ [10000 −15000] and $d^t_{rj} \in [5—10]$ with number of locations = 10, number of applications = 100, number of resources = 8 and number of periods = 8. These baseline values are somewhat higher than the corresponding values used to compare the historical, staged, and mixed models in Sections 4 and 5 in order to magnify the effects of varying parameter values. In each experimental description, we specify variations from these baseline values. To report results within a single graph in a concise manner, we normalize result values between the range [0,1].

6.1. Data center manager: effect of allocation–deallocation cost

We present the results of two experiments. In the first, we vary ranges for the allocation–deallocation cost of all resources. In the second, we vary the allocation–deallocation cost of a single resource.

In the first experiment, the range for the variable allocation–deallocation cost is varied from [0–10] to [400–410], and the ratio of fixed allocation–deallocation cost to variable allocation–deallocation cost is kept constant at 5. For each range of variable allocation–deallocation cost, we generate five problem instances and compute the average total cost ("Total Cost"), where the total cost refers to the objective function from the model, and the total number of allocations/deallocations ("#Alloc–Dealloc") across all problem instances. Because the ratio of fixed allocation–deallocation cost to variable allocation–deallocation cost is kept constant at 5 throughout all experiments, the increased variable allocation–deallocation cost will result in an overall increase in allocation–deallocation cost.

The results appear in Fig. 3. As the allocation–deallocation cost is increased, the number of allocations/deallocations decreases and the total cost increases. When the variable allocation–deallocation cost is in the range of [250–260], the number of allocations/deallocations reduces to the minimum value. Beyond this range, further increases in allocation–deallocation cost result in increased total cost; however, no allocation changes occur. At this point, allocations/deallocations occur to sustain the application demand for resources, not to reduce the total cost.

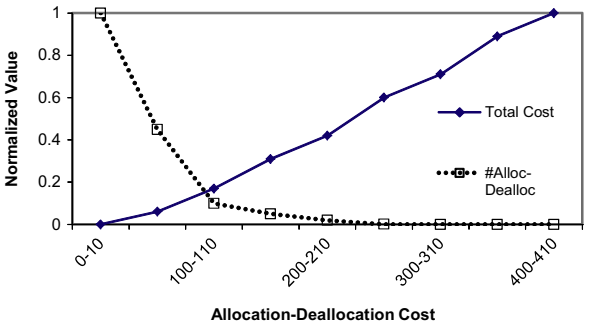

Fig. 3. Effects of allocation/deallocation cost.

In the second experiment, the variable allocation–deallocation cost for a single resource is increased from [0–10] to [60–70], and the ratio of fixed allocation/deallocation cost to variable allocation–deallocation cost is kept constant at 5. In Fig. 4, we plot how the cost associated with the resource ("Resource Cost") and the number of allocations/deallocations for that resource ("#Alloc–Dealloc") vary with the variable allocation–deallocation cost of the resource.

As the variable allocation–deallocation cost of the resource increases, the number of resource allocation–deallocation events decreases and reaches a minimum value, whereas the cost associated with the resource increases along with the allocation–deallocation cost.

At very high alllocation/deallocation costs, an allocation/deallocation event (if any) occurs only to support the application's resource demand, not to decrease the data center's cost. If a technology requires significant human interaction, and thus has a sizable allocation/deallocation cost, the cost savings from the deployment of virtualization technology may be nullified by the additional cost of maintaining it. IT managers can use this model to predict the total cost and determine the applicability of virtualization technology in their data centers from a cost perspective. We summarize the two boundary conditions of the allocation/deallocation costs as follows.

Insight 3: If the allocation and deallocation costs are high, the resource-to-application allocation remains unchanged unless there is a change in demand or other constraints on the resources.

This demonstrates the current scenario in most organizations, where virtualization technology is not used. Here, resources are allocated on the basis of peak demand and the application-to-resource binding is so tight that allocation changes are expensive. In these scenarios, unless the peak demand requires additional hardware or resources, the bindings are not altered. With middleware virtualization technology, however, deallocation and allocation are less expensive. This encourages the dynamic allocation and deallocation of resources, enabling companies to minimize the total infrastructure costs so that more applications can run on fewer resources.

Insight 4: If the allocation/deallocation cost is 0, the model becomes a resource allocation problem that can be solved dynamically within seconds, enabling integration with tools so that resources are automatically allocated to or deallocated from applications.

In an ideal virtualized environment, there are no costs associated with allocation and deallocation, and allocation and deallocation events take place purely based on the needs of the systems (similar to dynamic memory allocation schemes in modern operating systems). Present-day technology can reduce costs enough to make dynamic allocation and deallocation economically desirable, but not to reduce these costs to 0.

6.2. Application owner: effect of demand

In this experiment, we demonstrate how the demand for various resources by a single application $j = 1$, changes the application cost. We vary the demand of application $j = 1$ for various resources, d^t_{r1}. Demand values are selected from a set of ranges from [3–6] to [27–30]. For each of these ranges for d^t_{r1}, we generate five problem instances and compute the average of number of allocations/deallocations for the application ("#Alloc–Dealloc"), the cost of the application ("Application Cost"), and the cost of the application per unit resource ("Application Cost per Resource Unit"). We plot these against the ranges for d^t_{r1} in Fig. 5.

The cost of the application and the number of allocations/deallocations both increase almost linearly with the demand of the application. However, it is interesting to consider the variation of application cost per resource unit. Initially, until the range [15–18], the application cost per resource unit decreases due to increased utilization of resources, which reduces fixed costs per resource. However, as demand increases, the application is forced to acquire comparatively costly resources for running the application, resulting in increasing application cost per resource. Beyond the range [15–18], further increases in the application demand, using more costly resources, negate the decreases in cost related to improved utilization. This results in increased application cost per resource unit along with the demand from range [15–18] to [27–30].

Insight 5: As an application begins to make use of more expensive resources at higher demand levels, the fixed-cost savings accrued by improving the utilization of resources can be negated by the cost of using expensive resources.

6.3. Application owner: effect of budget

In this section, we vary the budget (B_j) of a single application $j = 1$ in the range $B_1 \in [5000 - 10000]$. For each value of B_1, we

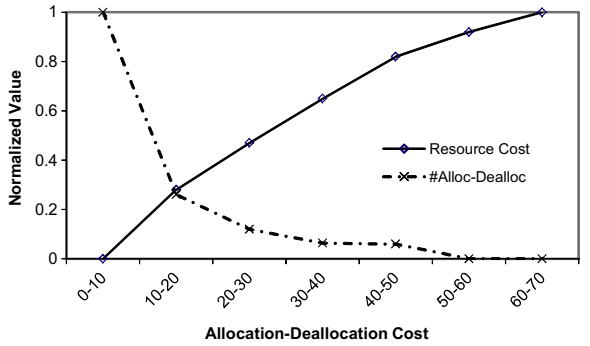

Fig. 4. Effects of allocation/deallocation cost for a single resource.

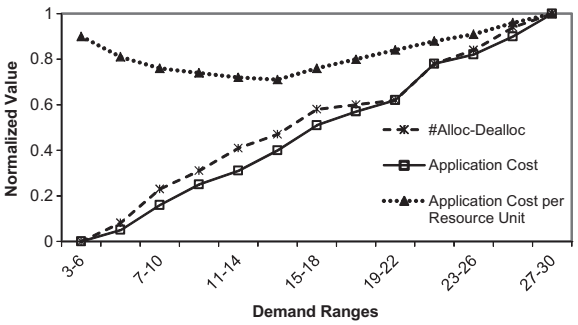

Fig. 5. Effect of demand for a single application.

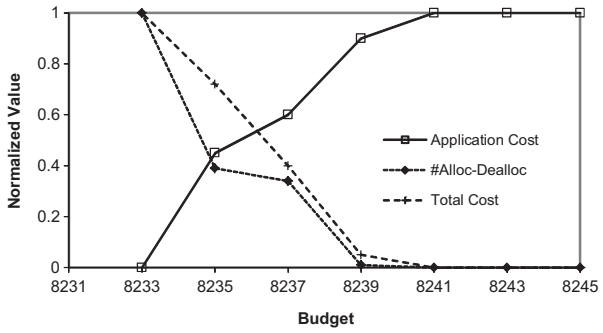

Fig. 6. Effect of budget for a single resource.

generate 5 problem instances and compute the average total cost ("Total Cost"), the number of allocations/deallocations ("#Alloc–Dealloc") and the application cost ("Application Cost"), i.e., the objective cost produced by the model. We plot the results in Fig. 6, showing the small range of budget values where variation occurs. As the budget of a single application increases, the total cost and the number of allocations/deallocations of the application decreases, while the cost of that particular application increases. This occurs because the application is moved to costlier resources while still remaining within the budget, while other applications are allowed to use less expensive resources. This will result in increased application cost for the single application, while the total cost of the system decreases. Effectively, the additional budget dollars allocated to the single application provide no benefit to that application; rather, they are used to reduce the costs of other applications and thus the overall cost of the system.

Insight 6: In situations where an IT manager is responsible for all application costs, the IT manager should allocate higher budgets for all applications – this will reduce the total cost of the system, as the model produces a cost-optimal allocation for the overall system. However, in situations where each application's budget is set and paid by an individual application owner, the application owner should be careful not set too high a budget – this will allow the cost of the application to increase with no incremental benefit to that application. Rather, the additional budget will be used to optimize the total cost of the system by reducing the expense incurred by other applications.

7. Summary and conclusions

In this paper, we developed a model to minimize the cost of running applications in an middleware virtualization framework for IT data centers. We modified the existing Dynamic Slope Scaling Procedure using historical load trends as a guide to future loads to develop an efficient solution technique for our model. We demonstrated the accuracy of our solution approach. To counter unforseen load we developed a staged approach, where resources are allocated based on observed actual loads. We demonstrated that the staged approach will result in about 20–30% higher costs than the near-optimal result. To combine the benefits of both the staged approach and the historical load based approach, we developed a mixed approach, and demonstrated that the mixed approach results in total cost within 7% of the near-optimal cost at moderate deviations from historical loads. In a set of sensitivity experiments, we show the impact of variation in allocation/deallocation costs, demand, and budget. Further, we include with our results a series of managerial insights that managers can use to decide which approach to use for resource allocations in a virtualized environment, including situations where the choice may vary based on stakeholder incentives.

Our work here is focused on the middleware virtualization context, and takes a first step toward incorporating operational factors such as salary, power, and variations across locations that are not traditionally considered in scheduling work. We further suggest that incorporating such factors in scheduling models represents a broad avenue for future research for the Operations Research community in general. We intend to continue pursuing this area of research in future work.

References

Aberdeen Group, 2008. Justifying the Cost of Uptime: Server and Storage Virtualization. <http://www.aberdeen.com/summary/report/benchmark/RA_VIRT_SL_3905.asp>.

Abramson, D., Giddy, J., Kotler, L., 2000. High performance modeling with nimrod/g: Killer application for the global grid? In: Proceedings of International Parallel and Distributed Processing Symposium.

Acey, M., 2006. Business Evolution. <http://business.timesonline.co.uk/article/0,,26012-1977360,00.html>.

Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y., 2008. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187 (3), 985–1032.

Amazon Web Services, 2010. Amazon Elastic Compute Cloud. interrefhttp://aws.amazon.com/ec2/url<http://aws.amazon.com/ec2/>.

Averbakh, I., 2010. On-line integrated production–distribution scheduling problems with capacitated deliveries. European Journal of Operational Research 200 (2), 377–384.

Bapna, R., Das, S., Garfinkel, R., Stallaert, J., 2008. A market design for grid computing. INFORMS Journal on Computing 20 (1), 100–111.

BEA Systems Inc., 2004. WebLogic Application Server. <http://www.bea.com/products/weblogic/index.html>.

Belady, C., 2007. In the data center, power and cooling costs more than the it equipment it supports. ElectronicsCooling. <http://www.electronics-cooling.com/2007/02/in-the-data-center-power-and-cool>.

Berman, F., Wolski, R., 1997. The apples project: A status report. In: Proceedings of 8th NEC Research Symposium.

Business Wire, 2007. The Infrastructure Virtualisation Market: Transforming the Way Information Technology (IT). Technical Report.

Buyya, R., Abramson, D., Giddy, J., Stockinger, H., 2002. Economic models for resource management and scheduling in grid computing. Concurrency and Computation: Practice and Experience 14 (13–15), 1507–1542.

Cai, X., Zhou, S., 1999. Stochastic scheduling on parallel machines subject to random breakdowns to minimize expected costs for earliness and tardy jobs. Operations Research 47 (3), 422–437.

Chen, Z.-L., Vairaktarakis, G.L., 2005. Integrated scheduling of production and distribution operations. Management Science 51 (4).

ComputerWorld, 2009. Salary Survey. <http://www.computerworld.com/s/article/9139190/Salary_Survey_2009>.

Hancock, S., 2009. Iceland Looks to Serve the World. <http://news.bbc.co.uk/2/hi/programmes/click_online/8297237.stm>.

Havill, J.T., Mao, W., 2008. Competitive online scheduling of perfectly malleable jobs with setup times. European Journal of Operational Research 187 (3), 1126–1142.

He, X., Sun, X., Laszewski, G., 2003. Qos guided min–min heuristic for grid task scheduling. Journal of Computer Science and Technology 18 (4), 442–451.

Herroelen, W., Reyck, B.D., Demeulemeester, E., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research 25 (4), 279–302.

IBM Inc., 2006a. Application Performance Evaluator and Resource Allocation Tool (APERA). <http://www.alphaworks.ibm.com/tech/apera>.

IBM Inc., 2006b. WebSphere Application Server. <http://www.ibm.com/websphere>.

ILOG Inc., 2009. ILOG CPLEX. <http://www.cplex.com/>.

Intel Inc., 2008. Data Center TCO: A Comparison of High-density and Low-density Spaces. <http://www.intel.com/technology/eep/datacenter.pdf>.

Kim, D., Pardalos, P., 1999. A solution approach to the fixed charge network flow problem using dynamic slope scaling procedure. Operations Research Letters 24 (4).

Kim, H.-J., Hooker, J., 2002. Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach. Annals of Operations Research 115, 95–124.

Kumar, S., Dutta, K., Mookerjee, V.S., 2009. Maximizing business value by optimal assignment of jobs to resources in grid computing. European Journal of Operational Research 194 (3), 856–872.

Kumar, S., Jacob, V.S., Sriskandarajah, C., 2006. Scheduling advertisements on a web page to maximize revenue. European Journal of Operational Research 173 (3), 1067–1089.

Kusnetzky, D., 2009. Virtualization Software Market to Top $10B by 2013. <http://www.zdnet.com/blog/virtualization/virtualization-software-market-to-top-10b-by-2013/1482>.

Lee, C., Lei, L., Pinedo, M., 1997. Current trends in deterministic scheduling. Annals of Operations Research 70, 1–41.

Litzkow, M., Livny, M., Mutka, M., 1998. Condor – A hunter of idle workstations. In: Proceedings of 8th International Conference of Distributed Computing Systems.

Liu, C., Yang, L., Goster, I., Angulo, D., 2002. Design and evaluation of a resource selection framework for grid applications. In: Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11).

Masiero, R., 2006. Moving from Arthritic IT to Dynamic IT: The Evolution of Datacenter. <http://it.sun.com/eventi/datacenter/pdf/Roberto_Masiero.pdf>.

McKinnon, W., Kallman, E., 1987. Mapping chargeback systems to organizational environments. MIS Quarterly 11 (1), 5–20.

Microsoft Inc., 2009. Microsoft Virtual Machine Solution for Application Server Consolidation and Migration. <http://www.microsoft.com/windowsserversystem/virtualserver/evaluation/vmnews>.

Min, R., Maheswaran, M., 2002. Scheduling co-reservations with priorities in grid computing system. In: Proceedings of 2nd IEEE/ACM International Symposium on Clsuter Computing and the Grid (CCGRID'02).

Mokotoff, E., 2001. Parallel machine scheduling problems: A survey. Asia-Pacific Journal of Operational Research 18 (2), 193–242.

Pearlson, K., Saunders, C., 2006. Managing and Using Information Systems, third ed. Wiley.

Pochet, Y., Vyve, M.V., 2004. A general heuristic for production planning problems. INFORMS Journal on Computing (3).

Polyakovsky, S., M'Hallah, R., 2009. An agent-based approach to the two-dimensional guillotine bin packing problem. European Journal of Operational Research 192 (3), 767–781.

Rackspace Hosting, Inc., 2010. Hosting Solutions. <http://www.rackspace.com>.

Ransbotham, S., Murthy, I., Mitra, S., Narasimhan, S., 2010. Sequential grid computing: Models and computational experiments. INFORMS Journal on Computing.

Ridouard, F., Richard, P., Martineau, P., 2008. On-line scheduling on a batch processing machine with unbounded batch size to minimize the makespan. European Journal of Operational Research 189 (3), 1327–1342.

Salary.com, 2010. Salary Wizard. <http://salary.com/>.

Santos, C., Zhu, X., Crowder, H., 2002. A Mathematical Optimization Approach for Resource Allocation in Large Scale Data Centers. Technical Report HPL-2002-64(R.1), HP Laboratories, Palo Alto. <http://www.hpl.hp.com/techreports/2002/HPL-2002-64R1.pdf>.

Scheier, R., 2007. Low-Cost Data Center Locations. <http://www.computerworld.com/s/article/300809/Low_Cost_Locations>.

Shao, G., Berman, F., Wolski, R., 2000. Master/slave computing in the grid. In: Proceedings of 9th Heterogeneous Computing Workshop.

Stone, B., Vance, A., 2010. Companies Slowly Join Cloud-Computing. <http://www.nytimes.com/2010/04/19/technology/19cloud.html?ref=global-home>.

US Energy Information Administration, 2010. Average Retail Price of Electricity to Ultimate Customers by End-Use Sector, by State (January 2010). <http://www.eia.doe.gov/electricity/epm/table5_6_a.html> (last accessed 15.04.10).

Vanover, R., 2007. SolutionBase: Bring Chargeback Management to Your VMware Environment. <http://techrepublic.com.com/2415-9592_11-170558.html>.

Vollmann, T., Berry, W., Whybark, D., Jacobs, F., Vollmann, T., Berry, W., 2004. Manufacturing Planning and Control Systems for Supply Chain Management: The Definitive Guide for Professionals. McGraw-Hill.

Wolski, R., Plank, J.S., Brevik, J., Bryan, T., 2001. Analyzing market-based resource allocation strategies for the computational grid. International Journal of High Performance Computing Applications 15 (3), 258–281.

Yao, J., Xiao, L., Nie, C., Wong, D.T.C., Chew, Y.H., 2008. Resource allocation for end-to-end qos provisioning in a hybrid wireless wcdma and wireline ip-based diffserv network. European Journal of Operational Research 191 (3), 1139–1160.