RC24478 (W0801-102) January 24, 2008
Computer Science

IBM Research Report

A Policy Evaluation Tool for Multi-Site Resource Management

Mudhakar Srivatsa, Nithya Rajamani, Murthy Devarakonda
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

A Policy Evaluation Tool for Multi-Site Resource Management

Mudhakar Srivatsa, Nithya Rajamani and Murthy Devarakonda
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
{msrivats, rnithya, mdev}@us.ibm.com

***Abstract*— An enterprise typically operates multiple datacenter sites, each handling workloads according to an enterprise-level strategy. Sharing resources across multiple sites (or enterprises) brings up several important problems. Each site may have its own policies that govern its interactions with other remote sites. Different policies impact the system performance in different ways. The site administrators and system designers need to understand the effects of a given set of policies on different workloads. In this paper, we describe an analysis methodology that determines the impact of policies on the workloads, and we present results and validation for a prototypical multi-site resource sharing system. Our analytical tool is capable of evaluating complex policies on a large scale system and permits independent policies for each site, so that policy makers can quickly evaluate several alternatives and their effects on the workloads before deploying them.**

***Index Terms*— Modeling and Prediction, Performance of Systems, Policy Impact Analysis, Resource Sharing, Distributed Systems**

I. Introduction

An enterprise typically operates multiple datacenter sites [2], [35], each handling workloads according to an enterprise-level strategy. The individual sites are federated and autonomous. Each workload has a Service Level Agreement (SLA) [36], in terms of its priority and performance requirements, and it caters to a specific customer base. Workloads are managed by workload managers, such as the IBM WebSphere Extended Deployment (XD Edition) [7], that are responsible for distributing individual requests of a workload among resources currently assigned to this workload. In order to facilitate resource sharing among the workloads at a site, a resource manager, such as IBM Tivoli Intelligent Orchestrator (TIO) [5], arbitrates resources among the workload managers at that site. Thus, a site resource manager is responsible for performing resource allocation across multiple workload managers at its local site.

The next level of resource sharing can happen across different sites of an enterprise. The resource manager at one site needs to coordinate with its counterparts at the remote sites for remote resource sharing. The enterprise can benefit from such multi-site resource sharing as underutilized resources at one site can be used to handle a peak load at a remote site, which would otherwise require over provisioning of resources at each site.

A variation of the multi-site enterprise scenario is the service provider scenario, where one site provides resources for a set of client sites. Yet another variation is where multiple sites operate on a peer-to-peer basis but do not belong to a single enterprise. Although the work described here is in the context of the multi-site single enterprise scenario, it is equally applicable to the other variations.

Sharing resources across multiple sites (or enterprises) brings up several important problems. In order to preserve the autonomy of each site, it is necessary that all decisions about the local resources should be made by the site resource manager. Hence, each site may have its own policies that govern its interactions with other remote sites, i.e. when, what, and how to share resources with the other sites. Different policies impact the SLA violations experienced by local and remote workloads in different ways. The site administrators and system designers need to understand the effects of a given set of policies on different workloads. We describe here an analysis methodology that determines the impact of policies on the workloads, and we present results and validation for a prototypical multi-site resource sharing system.

The results are useful for system designers in building effective solution strategies, and the methodology can be incorporated into a planning and system management tool that would permit policy-makers to understand, fine-tune and analyze the effect of policies on the system. Our tool is capable of evaluating complex policies on a large scale system and permits independent policies for each site, so that policy makers can quickly evaluate several alternatives and their effects on workloads before deploying them.

We have developed our methodology based on extensive work in the field of performance analysis [28], [37], [32] and load sharing in distributed systems [25], [13], [18]. The main solution methodology used by our tool is a combination of closed queuing network and finite state automaton. We model the system's state transitions as a non-deterministic finite state automaton. A site's policies are used to identify the set of valid system states and set of valid state transitions. These transitions are annotated with rate at which the transition occurs to construct a queuing network. The solution to the queuing network gives the steady state probability that the system remains in any particular state. The steady state solution is directly used to study the effect of these policies on workloads (e.g.: average case profit, worst case loss, etc).

However, queuing network based analysis is faced with a state space explosion problem, that is, the number of states in the queuing network may explode to large numbers making fast and scalable policy analysis a challenging problem. In this paper, we introduce the notion of near equivalent states and present a tuneable state space compression algorithm that simultaneously achieves the following goals: (i) reduce the number of states in the queuing network by several orders of magnitude, thereby facilitating fast and scalable policy analysis, (ii) retain the accuracy of steady state solution to the queuing network and thus preserve the integrity of overall cost analysis, and (iii) identify bottleneck resources in the system, and thus facilitate the system administrator to perform more intelligent capacity planning and policy tuning. We used our methodology to analyze

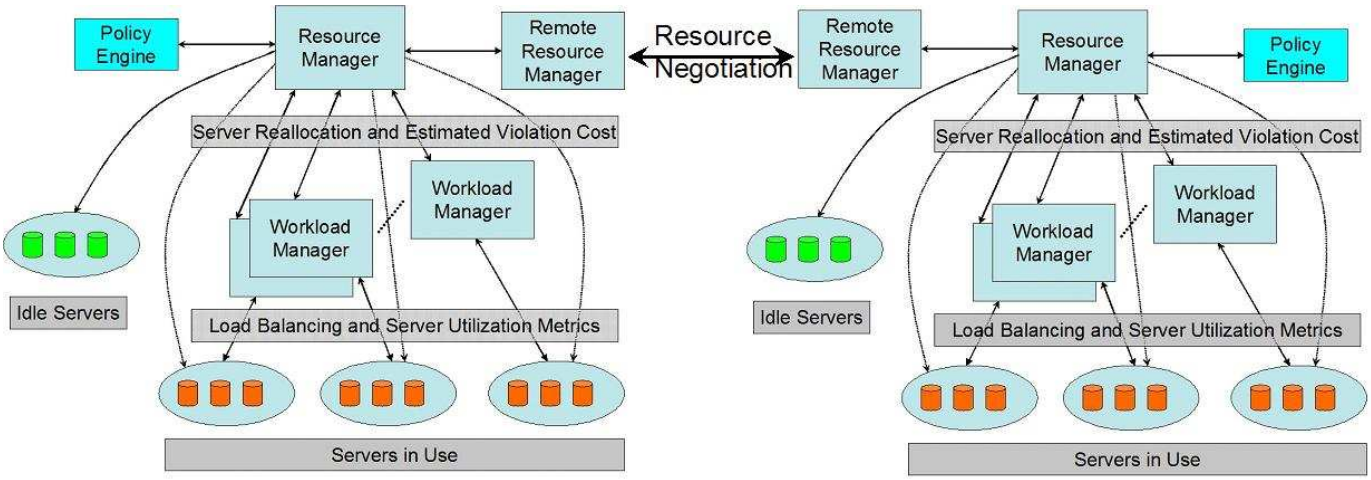

Fig. 1. Multi-Site Resource Management

several multi-site resource usage policies. The tool was useful not only in understanding the effects of various policies, but also in characterizing the conditions under which a policy would be useful. The key results from our analysis are as follows:

- Multi-site resource allocation is effective when individual sites are moderately loaded, but experience high variance in their loads, and only a small subset of sites are heavily loaded.
- When the individual sites experience moderate loads and high load variance greedy strategies work well.
- Allowing low priority workloads to borrow remote servers is useful, especially when the resource manager uses priority based preemption to allocate local resources.
- In using lease based resource sharing, long lease times hurt high priority workloads.

We validated our analysis methodology with a real-implementation of multi-site resource management system. Our prototype comprises of a simple two site scenario. It uses a benchmark J2EE application trade2 [8] as the workload, Cayuga [35] as the workload manager and TIO [5] as the resource manager. We observed that our analysis results match the measurements obtained from our implementation to within 5% error.

The following sections of this paper are organized as follows. We describe related work in Section II followed by an overview of multi-site resource allocation problem in Section III. We present a concrete model that captures the notion of site, workload, event, policy and cost in Section IV. We present algorithms for policy evaluation and combating state space explosion in Section V. Section VI describes experiments that quantify the scalability of our approach followed by a detailed collection of case studies. Section VII validates our policy evaluation tool against as real implementation. Finally, we conclude in Section VIII.

II. Related Work

Although load sharing in clusters and processor sharing [34], [29] are widely studied, a methodology for analyzing multi-site resource sharing under policy constraints is relatively new. The most closely related work is in the field of grid computing [1], [23], [22]. Open grid services architecture (OGSA [24], [21]) develops a framework (for both commercial and scientific grid) to support distributed system integration, virtualization and management services. Grid resource allocation manager (GRAM) [27] supports locating, submitting, monitoring and canceling jobs on a Grid.

Our work falls under the category of a commercial grid that is built on top of IBM Tivoli Intelligent Orchestrator (TIO) [5]. Platform Load Sharing Facility (LSF [10]) offers several infrastructural features similar to IBM TIO [5]. Platform LSF supports multi-cluster capabilities allowing a cluster to span across multiple sites across various geographical locations. Similar to TIO, LSF attempts to provide a single computing machinery image across multiple connected hardware clusters. LSF also supports rich multi-site resource allocation policies including job priorities and preserves local ownership and control. In this paper, we assume that the infrastructure required to support multi-site clusters is available (say, using IBM TIO or Platform LSF). Our goal is to develop fast and scalable algorithms for performance based policy impact analysis: *What if we use policy P instead of policy Q?*, *What if we change the threshold parameter in policy P from thr_1 to thr_2?*, *What if site A prefers sharing its resources with site B over site C?*, *What if the sites have different (and possibly conflicting policies)?*, etc. While Platform LSF supports a rich set of resource sharing policies and tools for monitoring resource utilization, it is not evident that it supports policy impact analysis. The algorithms described in this paper for fast and scalable policy evaluation (e.g.: cost analysis, what if analysis, etc) have been implemented using IBM TIO; but they may be applicable to several other multi-site resource management systems.

Several authors have addressed policy related issues in virtual organization (VOs) [18], [30], [31], [19] that may span multiple autonomous organizations. In [18], [19] the authors introduce usage-policies for resources in a grid and evaluate these policies using simulations and measured validation. The authors model virtual organizations that generate batch jobs to be executed at various resource provider sites. The resource providers accept jobs based on usage-policies. The study finds that a policy called commitment-limit is most effective in minimizing the response times. The commitment-limit policy specifies that a site should accept a job if a virtual organization's resource usage is below a threshold or if there are idle nodes and the virtual organization's use of resources is below another threshold. In [31], [19]

the authors extend policy based resource allocation techniques for hierarchical VOs. Our work is motivated by the need to study policy-based work offloading from one site to another, in a federated environment, using different system and workload models and considers a broad range of policies. Therefore, we studied a different aspect of policy-based resource sharing and thus providing a complimentary but different set of results. We believe our methodology is general enough that various usage-policies can be modeled and analyzed using our tool.

Significant amount of work has been done in the field of economic models for resource allocation in peer-to-peer networks. In [16], authors have proposed trading as a mechanism wherein a site acquires remote resources by trading away its own local resources. Several authors have also researched on optimal resource allocation in peer-to-peer networks [40], [17]. Our architecture for multi-site resource management is peer-to-peer. However, unlike peer-to-peer systems wherein the peers may be greedy and competing with each other, our focus is on cooperative multi-site resource management. Our work falls along with line of several studies that indicate that commercial sharing of IT resources yield significant advantages [14], [15].

Many papers describe the system architecture required to support cluster-based resource sharing and load balancing for web servers and application servers. Fox et al [25] describe the idea of using a separate overflow pool of servers (that are not usually a part of the cluster) for handling temporal bursts in the web traffic for one application. A report on giant-scale services [13] presents an extensive discussion on Internet-based systems that are primarily single-owner and comprise of well-connected clusters. The basic model of the giant-scale services implementation is similar to our site resource manager that attempts to hide node failures and balances traffic. However, the emphasis of the paper is on lessons learned from the infrastructural facilities available for effective load balancing and on enhancing the performance of application services such as round-robin DNS, layer-4 and layer-7 web switches. The emphasis here is on the evaluation of multi-site resource sharing policies in a system model of a few small enterprise data centers to the giant-scale services.

III. Multi-Site Resource Allocation

In this section we present the outline of our framework that could be employed by a resource manager to share resources with its counterparts. As we have already mentioned, it is very important to preserve the autonomy of each site. Hence, all decisions on the local resources available at that site should be made by the site's resource manager. Having a centralized resource manager that coordinates the resource managers at each local site might solve this problem in a very similar way that a site's resource manager coordinates various local application managers. However, the resource managers at each site would loose their autonomy over local resources. Hence, the resource managers at each site coordinate with their counterparts in a peer-to-peer manner. The fundamental primitives that are used by resource managers to interact with their counter-parts are: resource borrowing and resource donating.

A. Resource Borrowing and Donation

In our framework, each site is capable of borrowing and donating resources. When a resource is donated from site A to site B, site B can use that resource for time duration specified by lease time. At the end of the lease time, site B may request site A to renew the lease. Site A may renew the lease depending on its current state. Our resource borrowing and sharing differs radically from traditional resource models along two dimensions: resource granularity and time granularity. We assume that the resource granularity is one computing node in a server. There are two primary reasons for choosing this granularity level in a commercial grid. (i) Configuring one node to host multiple applications is challenging. (ii) More importantly, for security reasons (accidental information leakage), it may be unsafe to host two applications (especially if they are from different clients) on the same node. Having said that, as isolation and virtualization techniques [9], [11] improve it would be possible to use finer grained resources. Nonetheless, our policy evaluation tool can be easily extended to accommodate such fine grained resources.

The time granularity at which a resource is leased could vary from a couple of minutes to hours, as against a few milliseconds of CPU time slice. This large resource granularity can be attributed to the fact that datacenters (sites) typically own 10's or even 100's of nodes. The coarseness in time granularity is because it takes long to switch nodes among different applications; this includes setting up the required software stack and application, provisioning the application and finally enabling it [6].

For the sake of simplicity, we serialize all updates to the logical state of a resource manager due to resource borrow and donate operations. The logical state update changes the in-memory state model and adds log entries; these operations are typically fast and thus this keep the information lag small. Following the synchronized logical state update, a workflow is kicked off that executes the actual borrow or donate operation. If necessary this workflow also installs the required software stack, installs the application and sets appropriate environment variables. The actual state change (trigged by workflows) may take several seconds to a couple of minutes.

B. Policies

In view of the resource borrowing and donating model discussed above, a resource manager needs to make the following decisions:

- When to borrow a node?
- Where to borrow a node from?
- When to donate a node?
- How to handle lease renewal requests?

These decisions could be performance related (for example, Never borrow a node from a remote site when there are free nodes available at the local site) or administrative (for example, Site A should never donate/borrow a resource to/from site B). Our framework permits a wide variety of such policies to be deployed by the system. In order to preserve site autonomy we permit sites to specify their own policies. Our analytical tool is capable of studying the effect of these (possibly) heterogeneous policies on different workloads.

IV. Model

In this section we present a concrete model that describes various entities (physical and logical) in multi-site resource allocation including: a machine, a site, a workload, a policy, an event, and a cost model. We also describe a model to capture the state of physical entities including: site state and workload state.

A. Machine Model

A machine or a server is presented as a five-tuple $\langle nnodes, cpu, mem, disk, nw \rangle$ where $nnodes$ denotes the number of homogeneous nodes (blades) in a server. For each node we have: cpu denote the amount of computing power, mem denote the amount of main memory, $disk$ denote the disk access bandwidth and nw denote the networking bandwidth available. For scalability reasons, all resources are discretized using a least count. For example, the cpu resource is discretized using a least count of 100 MIPs (million instructions per second) and the network resource (nw) using a least count of 1Mbps.

B. Site Model

A site is represented by an array of $\langle ns, mm, pt \rangle$ where ns denotes the number of nodes with machine model mm and pool type pt. Each pool represents the software stack deployed on the node including operating system, network stack, application stack and etc. Each site has a policy set P that describes site specific policies. Policies conform to the policy model described later in this section. A site is associated with a list of workloads (or applications) that are hosted by the site that conforms to the workload model described below.

C. Workload Model

Workloads denote applications hosted by a site. Each workload is associated with a Service Level Agreement (SLA [36]) that explicitly specifies the guarantees provided by the service provider (site) and the cost of violating those guarantees. Hence the service provider is faced with the challenge of simultaneously meeting multiple SLAs. We assume that each workload draws its resources (nodes) from one exactly one server pool type. We use a capacity planner [4] to determine the number of nodes required to satisfy the workload's SLA for different values of mean workload request arrival rates. The capacity planner essentially permits us to discretize the resource requirement for a workload at any instance of time into $nlevel$ levels. Each level is in turn mapped to resource required to handle the workload satisfactorily (according to its SLA). Resource requirement is represented as an array of $\langle nnodes, mm, pt \rangle$ where $nnodes$ denotes the number of nodes of machine model mm and pool type pt. We use empirical measurements to determine the distribution of the amount of time a workload stays at level i. These measurements are primarily used to build a workload generator that best simulates the dynamics on a commercial grid. Popular choices for such distributions include the exponential distribution and Pareto distribution. The workload transits between levels as specified by a transition probability matrix tpm: $nlevels \times nlevels$; the entry $tpm(i, j)$ specifies the probability that a workload transits from level i to level j ($1 \leq i, j \leq nlevels$).

D. Workload State Model

A workload's state is characterized by $\langle lv, nl, nb \rangle$, where lv denotes the current workload level (as described in the workload model above), nl denotes the amount of local resources that are currently serving this workload and nb is an array (one element per remote site) that denotes the amount of resources borrowed on behalf of this workload. Note that these resources (local and remote) belong to the pool type required by the workload. A collection of resources (local and remote) is quantified by the three tuple $\langle nnodes, mm, pt \rangle$, where $nnodes$ denotes the number of nodes of machine model mm and pool type pt.

E. Site State Model

A site's state model captures the current state of a site. In contrast to the site model which describes static parameters associated with a site, the sites state model captures the time varying parameters associated with the site. A site's state is represented as $\langle ws, as, nd \rangle$, where ws is an array of local workload states, as is an array of active resources that not in maintenance mode, and nd denotes an array (one element per remote workload) that denotes the amount of resources donated to some remote workload.

F. Policy Model

In our framework policies aid a resource manager in responding to events. Let S denote the current state of the system. Let P be a collection of policies. Our policy engine is event-driven, that is, it is invoked only when some external event e occurs. The policy set P is used to guide the system to a collection of plausible new states that conforms to the policy set. More concretely, when the policy set P is invoked when current system state is S on some event e, it returns a collection $\{(S_1, p_1), (S_2, p_2), \cdots, (S_n, p_n)\}$, where S_i is a valid next state and p_i is the probability with which the system is recommended to transit to state S_i such that $\sum_{i=1}^{n} p_i = 1$. Note that if n equals one then, there is only one next state that conforms to the policy set P; and if $n > 1$ the system would choose its next state probabilistically from the set $\{S_i: 1 \leq i \leq n\}$.

More concretely, a policy P_1 can be a $\langle predicate \rangle$ or a three tuple $\langle predicate, e, action \rangle$, where $predicate$ is a predicate on the system state S. If $P_1 = \langle predicate \rangle$ then all states S such that $predicate(S)$ is false is precluded. If $P_1 = \langle predicate, e, action \rangle$ then for all states S such that $predicate(S)$ is true and when an event e occurs, the policy requires the system to perform an action $action$. In general, our policy specification requires a method $nextState(S, action)$ (could be any arbitrary function (Java method)) that returns the next state S' when the system performs an action $action$ from state S. An action could involve borrowing/donating resources, reallocating local resources among workloads, etc.

G. Event Model

Events represent those external changes that would require the system to reallocate or redistribute its resources amongst workloads in order to meet its business objectives. We focus on two major types of events in our framework: (i) a workload event occurs when a workload requirement moves from one level to another, and (ii) a node event occurs when a node fails (or is moved into maintenance mode) or when a node reinstated into the system on recovering from failure (or on maintenance completion). A node event also occurs when a node is borrowed or donated and when the lease on a borrowed node expires.

Given a state S, there are a set of events E that could potentially occur when the system is in state S. For each event $e \in E$, the probability distribution of the time for event e to fire is assumed to be known. For workload events the probability distribution of the time to next event is obtained from the workload model.

We assume that nodes fail (and recover) independently and the time of failure (and recover) is assumed to follow an exponential distribution or Weibull (bath tub) distribution.

H. Cost Model

We have developed a cost model to evaluate the effect of various policies on the system. In general our cost model permits cost functions that are arbitrary functions of the system's state and system's state transitions. In this section, we present a sample cost model that considers three important costs: (i) VC: violation cost that represents the cost of violating a workloads SLA (ii) RSC: remote node cost that represents the cost of using a remote resource (iii) RC: reallocation cost that represents the initial setup and provisioning cost for a workload. Additionally, one could include the cost of operating a node measured in terms of power (dominated by cooling costs), space, and man hours for maintenance.

First, we use $\alpha(S, w)$ to denote the violation cost for workload w when the system is in state S. A popular choice for estimating violation cost would be to make the violation cost proportional to $nds(S, w)$, where $nds(S, w)$ denotes the difference between the number of nodes required for workload w and the number of nodes actually allocated to workload w (local and remote nodes inclusive) in state S. In this case the violation cost would be expressed as penalty per deficit node per unit time in the workload's SLA.

Second, we use $\gamma(S, w)$ to denote the cost of using a remote node for workload w when the system is in state S. A common choice for estimating remote node cost would be to make it proportional to the $nbs(S, w)$, where $nbs(S, w)$ denotes the number of remote nodes borrowed for workload w in state S. In this case, the remote node cost would be expressed as penalty per remote node per unit time in the site's policy set. Additionally, one might choose to distinguish between nodes borrowed from different sites. In that case the remote node cost for different nodes would depend on the site from which those nodes were borrowed from.

Third, we use $\beta(S, S', w)$ to denote the reallocation cost for workload w, when the system makes a transition from state S to state S'. Reallocation cost would typically depend on at least the following two factors: (i) whether the reallocated node was idle or running some workload? (ii) Whether the reallocated node was local or remote with respect to the workload? (iii) What is cost of provisioning and setting up the workload? First, the reallocation cost would be higher if the node was previously running some workload (say, due to the time expended in simply shutting down that workload on the node); also such a cost might depend on the workload (if any) that was previously running on the node. Second, the reallocation cost for a local node is likely to be much smaller than that for a remote node. This is primarily because a local reallocation does not have go through agreements between different sites [4]. Support for automated negotiations across sites is provided by PANDA [26] and WS agreements [12]. Third and the most important component of reallocation is cost of provisioning and setting up the environment required for the workload on the reallocated node. This might involve additions to the node's software stack, starting the workload's runtime environment, setting up database connections, and etc.

I. Sample Policies

In this section, we discuss some samples policies that are permissible in our model. We present some policies that aid the system in responding to the following questions: (i) When to borrow a node? (ii) Where to borrow nodes from? (iii) When to donate a node? For the sake of simplicity in the discussion below, we assume that workloads are prioritized in decreasing order of their violation costs (violation costs are typically specified in dollars).

1) When to borrow a node?: Let an event e denote the fact that a workload w requirement has increased. Then borrow policy could be defined as a collection of policies shown below. Let A denote the site to which workload w belongs. P_1: Always allocate idle nodes in preference to other nodes. P_2: Always use local nodes in preference to remote nodes. P_3: If Site A has currently donated nodes for some remote workload rw, then preempt remote workload rw if its priority is smaller than workload w. P_4: If Site A had allocated nodes to some local workload lw, then preempt local workload lw if its priority is smaller than workload w. P_5: A workload w is eligible for using a remote node only if its priority is greater than a minimum threshold.

2) Whom to borrow a node from?: Let us suppose that a workload w has qualified to borrow a remote node. Then, the site from which it borrows a node could be determined by the following policy. Let nf_X denote the number of free nodes currently at site X. P_6: Borrow a node from that site that has the maximum number of free nodes.

3) When to donate a node?: Let us suppose that site X has received a request for donating a node. Then the site grants the request based on the following policies. Let nf_X denote the number of free nodes currently at site X. Let the number of requested nodes be r. P_7: Donate for r' nodes for workload w, where r' = min($nf_X - F$, r), where F denotes the minimum number of free nodes required after granting a donation request. P_8: Workload w is eligible for borrowing a node only if its priority is greater than a minimum threshold.

V. Analysis

In this section we present techniques to analyze a complex policy-driven system. We incorporate policies into the system by modeling the system dynamics as a finite state automaton. Then we superimpose a queuing network model on the automaton that labels state transitions in the automaton with probability distribution functions. We then solve the model using numerical techniques or simulations and use this solution to estimate workload costs (violation cost, remote node cost and reallocation cost).

A. Finite State Automaton

We model the dynamics of a complex policy-driven system as a non-deterministic finite state automaton. The automaton is constructed automatically from the individual site's policy set that is defined according to the policy model. States in the automaton correspond to the system states. State transitions are triggered by events. We use P to denote the collection of policy sets from all sites in the system. The collective policy set P determines how the system responds to events. More specifically, policies serve three critical purposes: (i) Determines whether a given state S is legally permissible or not. We call a state S illegal if it does not conform

to the policy set P. (ii) Determines whether a transition T between two states S and S' is permissible or not. We term a transition (response to an event) illegal if the event is not handled in a way that conforms to policy set P. (iii) Determines the probability that the system makes a transition T from state S to state S' in response to a given event e. We illustrate the role of policies in generating the finite state automaton description for the system using the following three examples.

- Policy p_1: "Site A never borrows a remote node for workload w". Given a state S, its legality can be tested (with respect to policy p_1) by ensuring that the workload state for w should indicate that nb (the number of nodes borrowed by workload w) is equal to zero.
- Policy p_2: "Site A borrows a remote node for workload w from a remote site that currently has the maximum number of idle nodes (of the same server pool type as that required for workload w)". Given a transition T from state S to state S', its legality can be tested (with respect to policy p_2) as follows. If workload w were to have borrowed a node in transition T, then the state S' should indicate that the borrowed node belongs to the site that has the maximum number of idle nodes in state S.
- Policy p_3: "Suppose site A has to borrow a remote node for workload w. Let nf_X denote the current number of idle nodes at site X. Then, site A borrows a node from site X with probability proportional to nf_X". Policies like p_3 permit policy makers to add randomization techniques that are popularly used as a heuristics for performance enhancement.

POLICY GUIDED STATE SPACE EXPLORATION(system modem SM, workload model WM, policy set P)
(1) Let E be the set of all events
(2) Start with some valid state S_0
(3) SS = $\{S_0\}$
(4) For every unexplored state $S \in SS$
(5) For every event $e \in E$
(6) For every state $S' \in P(S, e)$
(7) Add transition T: $S \rightarrow^e S'$
(8) SS = $SS \cup P(S, e)$
(9) Mark state S as explored
(10) Repeat steps 4-10 until there are no unexplored states

Fig. 2. Policy Guided State Space Exploration

We encode policies into the function $P(S, e)$ as discussed in the policy model. Recall that $P(S, e)$ determines how the system responds to event e when it is in state S. We now present a simple technique to construct the finite state automaton from the sites model and their policies. We use algorithm in Figure 2 that uses a policy-driven technique to construct the automaton. The algorithm above starts with some valid (or legal) state S_0 and explores the state space in a policy driven manner. This technique allows us to automatically generate the automaton from the site model and their policies. By construction, every state S in the automaton is legal and every transition T in the automaton is permissible. The probability of transition is handled in the queuing network model that is juxtaposed on this finite state automaton. We observed that this policy-guided state space exploration technique allows us to speed up automaton construction drastically. This is because, among all the combinatorial choices that could potentially represent a state, very few ($< 0.1\%$) of them actually conformed to our policy set. For instance, simple policies like "Do not starve a workload whenever an idle node (of the required pool-type) is locally available", "Use local node in preference to remote nodes", "Use priority based preemptive scheduling to manage local nodes" tend to drastically limit the number of states. Hence, pruning the state space using a policy-guided technique turned out highly beneficial.

B. Queuing Network Model

We superimpose a queuing network model on the finite state model to annotate state transitions with their probability distribution functions. A transition T: $S \rightarrow^e S'$ is labeled with a tuple $\langle f_e, pr \rangle$. The function f_e describes the probability distribution of the event e that causes the system to transit from state S to state S'. The probability pr denotes the probability that the system transits from state S to S' in response to event e. For node events, the function f_e is an exponential distribution; for workload events the function could either be an exponential distribution or a Pareto distribution. Exponential distribution is amenable to numerical analysis and thus provides fast (though crude) results; while Pareto distribution captures a more realistic bursty workload characteristics [37].

A solution to the above model gives us $pr(S)$ for all states S and $rate(T)$ for all transitions T, where $pr(S)$ denotes the probability (over an extended period of time) that the system is in state S and $rate(T)$ denotes the rate (over an extended period of time) at which the system makes a transition T. We use values to estimate workload costs as shown below (in terms of mean cost units per unit time):

$$\begin{aligned} VC &= \sum_S \sum_w \alpha(S, w) * pr(S) \\ RSC &= \sum_S \sum_w \gamma(S, w) * pr(S) \\ RC &= \sum_{T: S \rightarrow S'} \sum_w \beta(S, S, w) * rate(T) \end{aligned}$$

We compute $pr(S)$ and $rate(T)$ from the model as follows. If the workload events are exponentially distributed then we compute the stationary probability distribution using standard techniques to solve a Markov chain. The stationary probability distribution directly gives us $pr(S)$ for all states S. For every transition T: $S \rightarrow^e S'$, $rate(T) = pr(S) * rate(f_e)$, where $rate(f_e)$ denotes the rate of the exponential distribution f_e. We use a discrete event simulation [20] to solve the model when workload events follow a heavy-tailed Pareto distribution. We run the simulation for a substantially long period of time t_{sim}. In the course of this run we measure the amount of time expended by the system in any state S by $t(S)$, and use this information to compute $pr(S) = \frac{t(S)}{t_{sim}}$. Similarly, we measure the number of times the system transits from state S to state S' using a transition T by $n(T)$ and estimate $rate(T) = \frac{n(T)}{t_{sim}}$. Given $pr(S)$ for all states S and $rate(T)$ for all transitions T, the workload costs can be estimated as discussed above.

C. State Space Explosion Problem

Theoretically, solving the queuing network for the steady state model permits us to evaluate a set of policies. However, even for

a reasonably small system the number of possible system states tends to be exponentially large. This greatly limits the scalability of the policy evaluation tool. However, we recognize that most of these system states may not be important. In this section, we propose techniques to coalesce near equivalent states. There are two primary advantages in resorting to state coalition: (i) the number of states in the system significantly reduces, and (ii) at the end of state coalition, we are left with important states.

We say that two states S_1 and S_2 are nearly equivalent if $distance(S_1, S_2) < disthr$ and $|\frac{cost(S_1)}{cost(S_2)} - 1| < costthr$. The distance $distance(S_1, S_2)$ is defined as the Cartesian distance between the two state vectors S_1 and S_2. When we compute the distance we ensure that different elements of the state vectors are normalized. For example, let us suppose that all processing speeds are between 100 MIPs and 1000 MIPs. Then, we normalize all processing speeds to a range (0, 1) by mapping X MIPs to $\frac{X-100}{900}$. Finally, when we compute the Cartesian distance between two normalized state vectors, we permit different weights to be associated with different resources. For example, when the application is highly computing and memory intensive, we could weight the computing resource and the memory resource with weights w_{cpu} and $w_{mem} \gg w_{nw}$ and w_{disk}.

However, the fact that $distance(S_1, S_2)$ is very small does not necessarily imply that the states can be coalesced. For instance, consider a state S_1 wherein the system utilization is 100% and a state S_2 which is very close to S_1 wherein the system is overloaded (and thus violating the SLA for some workloads). Although $distance(S_1, S_2)$ is very small the two states are not equivalent. So we add the second cost constraint that requires that the cost function at both the states S_1 and S_2 evaluate to nearly the same value. Cost of a state S for our sample cost model is determined as $cost(S) = \sum_w \alpha(S, w) * pr(S) + \sum_w \gamma(S, w) * pr(S) + \sum_{S'} \sum_w \beta(S, S, w) * rate(S \rightarrow S')$.

Note that in general a policy may specify arbitrary actions to events. Hence, strictly speaking the policy may require the system to react in entirely different ways on the same event e from states S_1 and S_2. Hence, one would have to compare all the transitions in and out of states S_1 and S_2 before we conclude that the states are equivalent. However, we believe that a consistent set of policies would not make radically different decisions on the same event e from two states S_1 and S_2 with a very small distance and comparable costs. Hence, our policy evaluation tool does not consider the transitions in and out of a state when determining the equivalence between two states. However, one can optionally turn checks on transition; then the evaluation tool would report possible inconsistencies in the policy set P. When two states S_1 and S_2 are coalesced, we replace them with a state S_{12} with $pr(S_{12}) = pr(S_1) + pr(S_2)$. For every transition T_1: $S_1 \rightarrow^e S$ and T_2: $S_2 \rightarrow^e S$ (for some state S), we replace with a transition T: $S_{12} \rightarrow^e S$ with $pr(T) = pr(T_1) + pr(T_2)$. The same holds for transitions T_1: $S \rightarrow^e S_1$ and T_2: $S \rightarrow^e S_2$ to be replaced with a transition T: $S \rightarrow^e S_{12}$ with $pr(T) = pr(T_1) + pr(T_2)$.

COALESCE AND SOLVE(State Space SS)
(1) Initialize $pr(S_j)$ using simulation
(2) Coalesce states based on distance and cost threshold to obtain a state set SS
(3) For i = 1 to $|SS|$
(4) Pick a random j such that $a_{ij} \neq 0$
(5) Compute $pr(S_j) = -\sum_{k \neq j} \frac{a_{ik} * pr(S_k)}{a_{ij}}$

Fig. 3. Coalesce and Solve

However, observe that $cost(S)$ depends on $pr(S)$ and $rate(T)$ (for all T: $S \rightarrow S'$ and T: $S' \rightarrow S$), which can be obtained only on solving the queuing network model for its steady state solution. We merge the process of coalescing states and solving the queuing network model as follows. Let the set of linear equations obtained from the steady state solution to the queuing network model be represented as $\sum_j a_{ij} * pr(S_j) = 0$ (for all i) and $\sum_j pr(S_j) = 1$. We start with an initial guess for $pr(S_j)$ using simulations (for faster convergence). Then based on the distance threshold and the cost threshold we iteratively coalesces states and solve for the steady state equation as shown in algorithm 3. In general, one can use many stopping conditions for the above algorithm. We have used a precision based stopping condition in the algorithm shown above. Alternatively, one can use a condition on the number of states ($|SS| < thr$), the running time of the policy evaluation tool ($runTime < thr$) or the total number of iterations ($numItr < thr$).

When the policy evaluation tool terminates, we are left with important states and only relevant components in the state vector. For example, consider a compute intensive application where disk i/o is irrelevant. Let us suppose that normalized network resource had initially four classes (0, 32Kbps), (32Kbps, 64Kbps), (64Kbps, 96Kbps) and (96Kbps, 128Kbps). At the end of the algorithm described above, we will be left with states wherein the network resource has only one class (0, 128Kbps). When any resource spans its full range, the resource becomes irrelevant (irrespective of the quantity of that resource the system behaves identical). Hence, we replace such resources with a special class '*' meaning that the resource is not important for the given application, the policy set and the cost function. In general we start with an arbitrarily large number (millions) of highly fine grained states. Our algorithm not only provides a steady state solution to the queuing network model, but also reduces the number of states to only a few important ones (a few tens depending on the distance and cost thresholds).

D. Unified Policy Evaluation Algorithm

In this section, we present our complete algorithm for policy evaluation. The algorithm takes a system model SM, a workload model WM, a cost model CM and a policy set P as its input. The output of the algorithm is a probability distribution function of the overall system cost, namely, $Pr(cost = x)$ for all x, where $Pr(cost = x)$ denotes the probability that the overall system cost is equal to x. Note that given the overall cost distribution, one could easily measure the average cost and its higher order moments (like standard deviation). The unified policy evaluation algorithm is shown in algorithm 4. We use a policy guided state space generation technique to generate the state space (algorithm 2). We then coalesce states depending on the distance and cost thresholds and solve the coalesced state space (CSS) for a steady state solution. The steady state solution gives the $pr(S)$, the probability that the system is in some state S for all $S \in CSS$. Now, we obtain the cost of each state S using a cost model and translate the probability distribution over the state space to a probability distribution over the overall system cost. The cost distribution allows the administrator to perform worst case analysis. For example, there could be two policy sets P and Q such that the average cost of P is lower than that of Q.

UNIFIED POLICY EVALUATION(system model SM, workload model WM, cost model CM, policy set P)
(1) SS = policy_guided_state_space(SM, WM, P)
(2) CSS = coalesce_and_solve(SS)
(3) CD = measure_cost(CSS, CM)

Fig. 4. Unified Policy Evaluation

However, P might have a state S, where $pr(S) > 0$ and $cost(S)$ is greater than the cost of all states in the state space of policy Q. The administrator can make such observation by looking at the probability distribution plots for the overall system for policy sets P and Q. This enables the administrator to make much sounder decisions about which policy set to choose for the system.

The unified policy evaluation algorithm also permits the administrator to carry out sensitivity analysis effectively. We study techniques to perform sensitivity analysis with respect to the system model, the workload model and the cost model. Let us suppose that we have evaluated a system model SM, workload model WM, cost model CM under the policy set P to obtain CSS (coalesced states). Let us now suppose that we make a small change $\triangle SM$ to the system model. Recall that SM is a state vector and $\triangle SM$ denotes the change to the system model and is also represented as a huge state vector such that the new system model SM' = SM + $\triangle SM$ (vector addition). We use $\triangle SM$ and update every state S in CSS to obtain S' = S + $\triangle SMS$, where $\triangle SMS$ denotes a projection of $\triangle SM$ on S. For example, if S has a $*$ on the disk i/o component in its vector then the disk i/o component in $\triangle SMS$ is also replaced by *; if $\triangle SM$ involved a machine with 300 MIPs replaced by a machine with 600 MIPs and the state S had its CPU resource vector marked with the range (0, 1000) MIPs then $\triangle SMS$ is (0, 1000) MIPs. Clearly, a project of $\triangle SM$ on S eliminates all changes that are irrelevant to S. Finally, we compute important changes to the system model with respect to the policy set S as imp_change = $\sum_{SS \in CSS} distance(S, S')$. If imp_change is larger than a threshold we solve the system afresh using algorithm 4; else we simply construct the new set of coalesced states CSS' by replacing every state S in CSS by S'. We use CSS' to obtain the cost distribution. We use the same technique described above for sensitivity analysis towards the workload model. However, one cannot use the same technique for the cost model since states may be coalesced using a cost threshold. We therefore start with the uncoalesced state space SS and evaluate the $cost(S)$ for every S in SS using the cost model CM and cost model CM'. If the mean difference $SS \in CSS$ $\frac{|cost_{CM}(S) - cost_{CM'}(S)|}{|SS|}$ is lesser than a threshold then we assume that this change in the cost model does not change the set of coalesced states. In this case, we use the same CSS to revaluate the cost distribution (step 3, algorithm 4); else, we rerun algorithm 4 from step 2.

Site Type	Workloads	Local Nodes
ST_1	$\{WT_1\}$	$\{\langle 3, 0\rangle\}$
ST_2	$\{WT_2\}$	$\{\langle 1, 0\rangle\}$
ST_3	$\{WT_2, WT_2\}$	$\{\langle 2, 0\rangle\}$
ST_4	$\{WT_2\}$	$\{\langle 2, 0\rangle\}$
ST_5	$\{WT_1\}$	$\{\langle 2, 0\rangle\}$

Fig. 5. Site Types

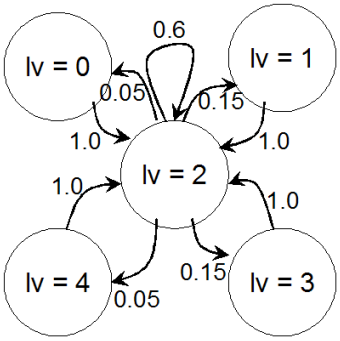

Fig. 6. Workload Model: Sample Transition Probability Transition Matrix

VI. RESULTS

In this section, we present several results obtained using our analytical tool to study various policies for multi-site resource management. For every experiment we used a different scenario that best highlights the inferences we made from them. A scenario is described using the site and workload model used to perform the experiment. We use a small set of site types and workload types in all our experiments. We first describe them in Figure 5 and 6. For example, Figure 5 shows that a site of type ST_1 runs a workload of type WT_1 and has three local nodes of pool-type zero; and a site of type ST_2 runs a workload of type WT_2 and has one local node of pool-type zero; and a site of type ST_3 runs two workloads both of type WT_2 and has two local nodes of pool-type zero. The workload types WT_1 and WT_2 are described later in this section.

We now describe the first workload type WT_1. A workload of type WT_1 has 5 levels numbered 0, 1, 2, 3, and 4. At level i, the workload requires i nodes. At each level i, the workload spends an exponentially distributed amount of time with a mean of 100 time units. The default transition probability matrix of the workloads is as shown in Figure 6. In our experiments, we vary this matrix to change the mean load and load variance. In this example, the mean load is about two nodes; when averaged over a long period of time, the workload is in level 2 for about 66% of the time, in levels 1 and 3 each for about 13% of the time and in levels 0 and 4 each for about 4% of the time. A workload of type WT_2 has 3 levels numbered 0, 1 and 2. At level i, the workload requires i nodes. At each level i, the workload spends a Pareto distributed amount of time with a mean of 100 time units and infinite variance. In our experiments we use different transition probability matrices to achieve different mean loads and load variances.

For all workloads we used the following simplified cost model. We assume that every workload w has a priority denoted by $priority(w)$. The violation cost parameter $\alpha(S, w)$ = $nds(S, w) * priority(w)$, where $nds(S, w)$ denotes the difference between the number of nodes required for workload w and the number of nodes actually allocated to workload w (local and remote nodes inclusive) in state S. The remote node cost parameter $\gamma(S, w)$ = 0, that is, there is no penalty for using a remote node. The reallocation cost $\beta(S, S, w)$ is defined to be equal to the violation cost experienced by the workload during the reallocation process. Based on our measurements on our prototype (see next Section), we observed that a reallocation involving a local node took 2 time units and that involving a remote node took 12 time units. Hence,

$\beta(S, S, w) = nlts(S, S', w) * priority(w) * 2 + nrts(S, S', w) * priority(w) * 12$, where $nlts(S, S', w)$ denotes the number of nodes involved in local reallocation and $nrts(S, S', w)$ denotes the number of nodes involved in a remote reallocation (for workload w when the system transits from state S to state S').

A. Scalability Experiments

In this section, we present performance and scalability results on our policy evaluation tool. First we study the effect of distance and cost threshold on the performance of the policy evaluation tool. Second, we show the ability of our evaluation tool to scale with the number of sites. Figure 7 shows the fraction of remaining states for different values of distance and cost threshold. Note that only important states are left behind when our policy evaluation tool terminates. Also, the higher the distance and cost threshold, more states would be coalesced. Note that the number of important states drops steeply as the threshold values are increased. This is primarily because most of the system states are indeed equivalent to one another and can be coalesced. For example, in a transactional grid, the cost model is independent of the disk i/o utilization. Hence, in all the final states disk i/o part of the state vector would be eliminated (replaced by *).

Figure 8 shows the accuracy of the evaluation tool for different values of distance and cost threshold. Accuracy is measured as the ratio of the estimated system cost with state coalition and without coalition. From Figure 7, as the threshold is increased the number of remaining states decreases sharply. However, the accuracy of the evaluation only marginally falls with the distance and the cost threshold.

Figure 9 shows the time taken for policy evaluation for different values of distance and cost threshold. The time it takes for the policy evaluation tool to terminate falls very sharply with the threshold values. This is primarily because of the reduced number of states. Note that the number of equations to be solved in order to obtain a steady state solution for the queuing network model is proportional to the number of states. Note that the cost of solving a system of n linear equations is proportional to n^2.

Figure 10 shows the scalability of the system with the number of sites for certain values of distance and cost threshold. Note when the threshold values are zero, the time it takes for the policy evaluation tool to terminate almost increases exponentially with the number of sites. This is because the number of possible system states increase exponentially with the number of sites, thereby severely limiting the scalability of the policy evaluation tool. However, as we raise the threshold the policy evaluation tool is much better equipped to handle a system with larger number of states.

B. Case Study

In this section we present a collection of case studies on multi-site resource allocation. Even though our policy evaluation tool allows different sites to use different cost models, in this section we assume that all sites use the same cost model.

1) When is Multi-Site Resource Allocation Useful?: In this experiment, we identify the workload characteristics that would make multi-site resource allocation a better choice when compared to independent non-cooperating sites. This comparison is achieved by explicitly comparing the aggregate violation cost of cooperating Vs non-cooperating sites. The workload characteristics of primary interest to us are the mean load and the load variance.

Scenario I: Two sites S_1 and S_2 both of type ST_1. Site S_1 has one workload W_1 of type WT_1. Site S_2 has one workload W_2 of type WT_1. Both the workloads W_1 and W_2 have the same priority.

Figure 11 shows the violation cost as the total mean load of workloads W_1 and W_2 varies (under fixed variance = 1). 'lvc' denotes the violation costs when the sites operate without cooperating with each other (they optimize resource allocations locally). 'vc' denotes the violation costs when the sites cooperate with another and borrow/donate nodes to handle peak loads. The Figure 11 can be divided into three zones: (i) At light loads, there is not much need to borrow resources and hence cooperating multiple sites do not succeed in reducing the aggregate violation cost significantly. (ii) At heavy loads the system (the collective resources available at all sites) is insufficient and hence, cooperation does not yield significant gains (unless the workloads vary largely in terms of their priorities). (iii) At moderate loads sites can offload their peak demands to free nodes available at remote sites thereby achieving much lower violation costs.

Figure 12 shows the aggregate violation cost as the load variance changes (at fixed mean load = 4). As the variance increases, the workloads spend most of their time at a state where they need 4 nodes or at a state where they need just 0 or 1 node. At lower load variance, the workloads spend a significant portion of their time close to their mean, that is, when the workloads each need 2 nodes. As the variance increases, cooperating multi-sites can handle peak demands at one site by borrowing resources from the other site; although the variance is high, the peak demands at the two sites are likely to be uncorrelated. Further, as the variance increases, the workloads spend more and more time units at a state where they require 4 nodes and at a state where they require 0 nodes. Hence, as the variance increases, cooperating multi-sites become a much better choice for resource allocation when compared to non-cooperating sites.

Scenario II: Two sites S_1 and S_2 both of type ST_5. Site S_1 has one workload W_1 of type WT_1. Site S_2 has one workload W_2 of type WT_1. Both the workloads W_1 and W_2 have the same priority.

In this experiment we demonstrate the usefulness of multi-site resource allocation when the sites are unevenly loaded. Figure 13 shows the aggregate violation costs when the sites are unevenly loaded. We fix the mean load on site S_1 to be 1.5 nodes and vary the mean load on site S_2. When the load on site S_2 is very low then cooperating multi-sites do not have any advantage. However as the load on S_2 increases, site S_2 can offload some its load to site S_1. But, when site S_1 gets loaded to its maximum capacity, it can no longer accept load from site S_2. Hence, the difference between 'lvc' and 'vc' does not diverge when the workload on site S_2 soaks up all the resources available in site S_2 and the unused resources in site S_1.

Figure 14 shows how our evaluation tool can be used to perform sensitivity analysis. Figure 14 shows a simple sensitivity analysis on the overall system cost as we vary the workload model parameters. We vary the mean load and the load variance (keeping the mean load a constant) and study its effect on the system cost. On the x-axis we show the factor by which a workload parameter is changed and the y-axis shows the corresponding factor by which the overall system cost changes. Figure 14 shows that keeping the mean load a constant, increasing the variance by

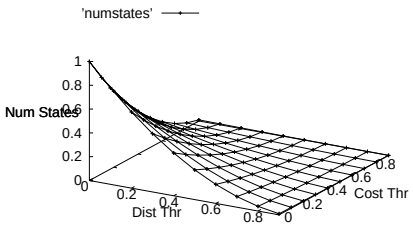

Fig. 7. Fraction of Remaining States

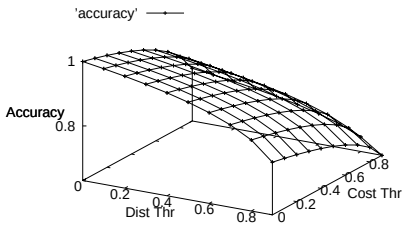

Fig. 8. Accuracy of Policy Evaluation Tool

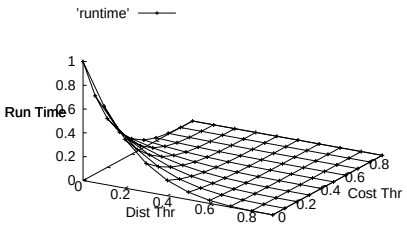

Fig. 9. Run Time

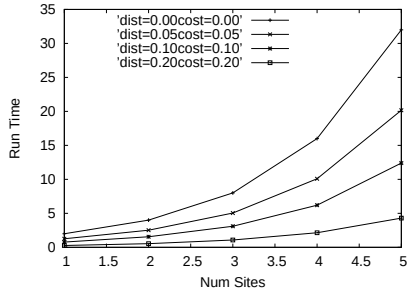

Fig. 10. Scalability with Number of Sites

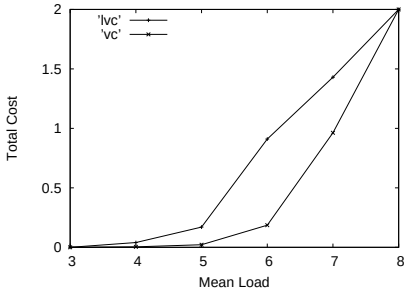

Fig. 11. Violation Cost Vs Mean Load

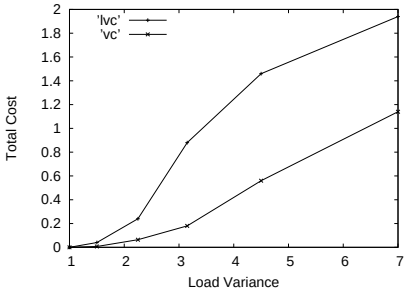

Fig. 12. Violation Cost Vs Load Variance

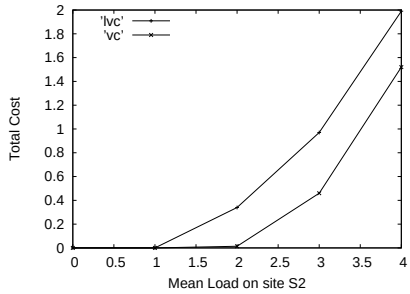

Fig. 13. Unevenly Loaded Sites

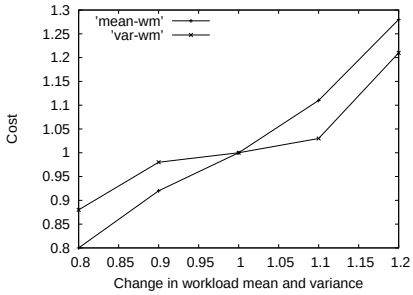

Fig. 14. Sensitivity Analysis

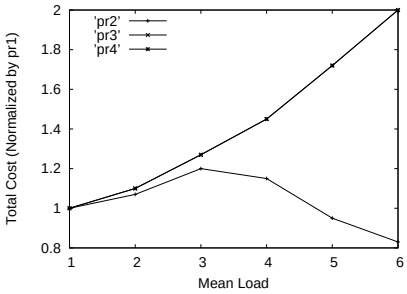

Fig. 15. Effect of Borrow Threshold

20% (a factor of 1.2), the overall cost goes up to the same extent as increasing the mean load by 12-14%.

2) Borrowing Remote Nodes: In this section we study the effect of borrow threshold thr_br on the aggregate violation cost. Note that when a borrow threshold is enforced, only workloads w with $priority(w) \geq thr\_br$ are permitted to borrow remote nodes. The first experiment on borrow threshold shows that borrow threshold is useful only when the system (inclusive of all sites) is operating at high mean load. The second experiment shows how our analytical tool can be used to perform worst case analysis.

Scenario III: Four sites S_1, S_2, S_3 and S_4 all of which are of type ST_2. Site S_i has one workload W_i of type WT_2 for $1 \leq i \leq 4$. Priority of workload W_i is i ($1 \leq i \leq 4$).

Figure 15 shows the aggregate violation cost Vs aggregate mean load for different values of borrow threshold. 'pr_j' indicates only workloads $\{W_i: i \geq j\}$ can borrow remote nodes. The values shown in the figure are normalized by 'pr_1' which indicates the violation cost when all workloads can borrow remote nodes. At very low loads, there is no need to borrow nodes; and hence the threshold has no effect on violation cost. As the mean load increases, there is opportunity to offload peak demand by borrowing remote nodes; setting a threshold inhibits the system from exploiting the free nodes available at remote sites for lower priority workloads. However, at very high load, using a borrow threshold is very useful; since the lower priority workloads are not allowed to borrow nodes, there is not much time wasted in reallocating resources among different workloads.

Figure 16 shows the cost distribution of two policies P_1 and P_2 under heavy mean load = 8. P_1 uses a borrow threshold of 4 and P_2 uses a borrow threshold of 2. Observe from figure 15 that the average cost of P_1 is lower than the average cost of P_2. However, the cost distribution in figure 16 shows that the worst case cost for P_1 is higher than the worst case cost of P_2. This is primarily because under policy P_1 three workloads are never permitted to borrow resources and thus it incurs a higher cost when the load due to workload W_4 is low and the rest (W_1, W_2 and W_3) are high. If the system administrator is interested in worst case costs, then the administrator can graphically view the cost distributions before deciding on the appropriate policy.

The second experiment on borrow threshold shows the effect of local resource allocation strategy. In this scenario, the resource manager uses a priority-based preemptive resource allocation strategy for managing local resources. Figure 17 shows the violation cost Vs mean load for different values of borrow threshold.

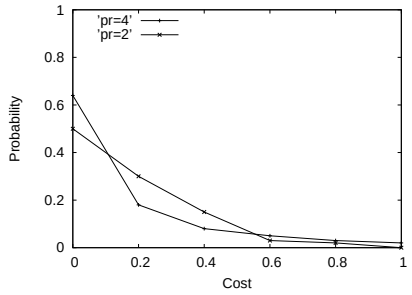

Fig. 16. Worst Case Analysis

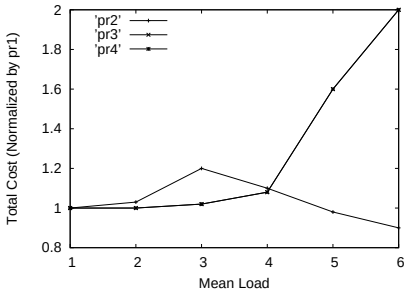

Fig. 17. Effect of Local Optimization Strategy

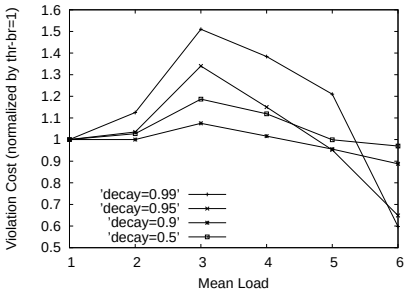

Fig. 18. Adaptive Borrow Threshold

'pr_j' indicates only workloads $\{W_i$: $i \geq j\}$ can borrow remote nodes. The values shown in the figure are normalized by 'pr_1' which indicates the violation cost when all workloads can borrow remote nodes. The main emphasis in this experiment is that one needs to be careful in choosing borrow thresholds. For instance, 'pr_3' and 'pr_4' in the figure behave much worse than 'pr_1' under all values of mean loads (even under high loads, as against figure 15). This is because, in this scenario, the workloads W_3 and W_4 never need to borrow nodes. The local optimization cycle always grabs node from a local lower priority workload and transfers it to a higher priority workload. Since, W_3 and W_4 require no more than two nodes, they are always guaranteed to be allocated local nodes. Hence, 'pr_3' and 'pr_4' are equivalent to the case where the sites are non-cooperating (W_3 and W_4 never need to borrow nodes; W_1 and W_2 are not permitted to borrow nodes).

The third experiment on borrow threshold shows the effect of threshold adaptation. In threshold adaptation, the threshold value is decayed by a constant decay factor on every remote optimization cycle. However, it is reset to its original (default) value whenever a borrow operation fails to obtain a remote node. When the system is heavily loaded, the borrow requests for lower priority workloads is very likely to fail and hence the borrow threshold would stay close to its default value. On the other hand, if the system is lightly loaded most borrow requests would succeed in fetching a free remote node. Therefore, at low loads, the borrow threshold would be very low and thus, most workloads would be permitted to borrow nodes. We use the same scenario as in Scenario III described earlier in this section. Figure 18 shows violation cost Vs mean load for different values of decay factor. At very low loads, the decay factor has no influence on violation cost since the individual sites have sufficient resources to handle their peak demands. When the decay factor is very close to one, we are being highly conservative in permitting lower priority workloads in borrowing nodes; thus, higher decay factors tend to perform poorly at moderate loads (underutilized resources). When the decay factor is low, we encourage lower priority workloads to borrow nodes; thus, low decay factors tend to perform poorly at high loads (thrashing due to frequent reallocation).

3) Cycle Breaking Rule: Scenario V: Four sites S_1, S_2, S_3 and S_4 all of which are of type ST_2. Site S_i has one workload W_i of type WT_2 for $1 \leq i \leq 4$. Priority of workload W_i is i ($1 \leq i \leq 4$).

Cycle breaking rule is a policy added to improve the systems stability: "No local workload with higher priority is executed on a remote node when a remote workload with lower priority is being run on a local node". The key motivation behind a cycle breaking rule is as follows. Let $A \rightarrow B$ denote that site A is using some nodes from site B. Let the set wp_A denote the priority of a workload that belong to site A and are currently running on nodes in site B (and similarly for wp_B). Then, the cycle breaking rule requires that $wp_A > wp_B$. Clearly, if the cycle breaking rule were strictly implemented then there would be no cycles in the resource borrowing graph, that is, site A will not be using the resources at site B and site B using the resources at site A at the same time instant. On the contrary, if we assume that there exists a cycle $A \rightarrow B \rightarrow \cdots \rightarrow A$ then it would mean that $wp_A > wp_B > \cdots > wp_A$ – an obvious contradiction.

Figure 19 compares the remote node cost ('rsc') and the violation cost ('vc') with and without cycle breaking rule for different values of mean load. The figure shows the ratio of each cost with cycle breaking rule to that without the rule. At very low load, no borrow operations are required as the sites are self-sufficient. As the mean load increases more and more nodes may be required to be borrowed. The cycle breaking rule decreases the number of borrow operations and thus ensures that the remote node cost and the reallocation cost are substantially smaller. At high load there are no free nodes available at either site that could be borrowed. Hence, at very high load the number of borrow operations come down and consequently the cost difference between a policy with and without cycle breaking rule decreases.

4) Lease Time: Scenario VI: Two sites S_1 and S_2 each of which is of type ST_4. Site S_1 has one workload W_1 of type WT_2 and site S_2 has one workload W_2 of type WT_2. Priority of workload W_i is i ($1 \leq i \leq 2$).

This section studies the effect of lease time on the workload violation costs. Lease time based policies allow resources to be borrowed or donated for a fixed period of time, namely, the lease time. Leases are non-preemptable, that is, once a resource is leased the donor cannot withdraw that resource before the specified lease time. On the other hand, the site that borrowed a resource could return the resource before its lease terminates.

Figure 20 shows the violation cost of the two workloads as the lease time increases. Note that leases are non-preemptable; but a borrow resource can be returned before a lease expires (if the borrowing site decides that the borrowed node is no longer required). Consider a scenario wherein the lower priority workload W_1 has borrowed a node. Now, if the higher priority workload W_2 needs a node, it has to wait till the lease expires. When the lease expires, W_1 is denied a lease extension and the node is assigned to W_2. Hence, as the lease duration increases, the violation cost for higher priority workload increases. Beyond a certain value, increase in lease duration does not affect the violation cost because the borrowing site would return the node (because its peak demand is over) almost always before the lease

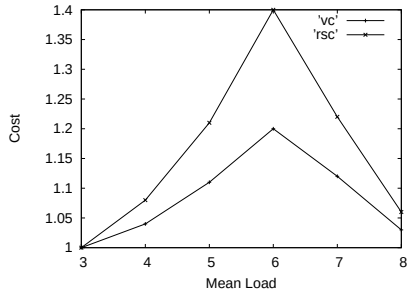

Fig. 19. Cycle Breaking Rule

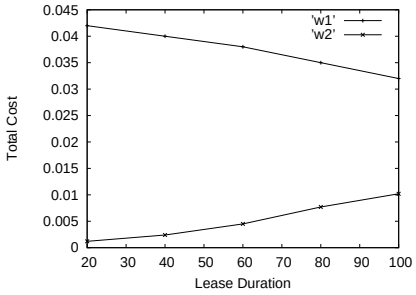

Fig. 20. Lease Time

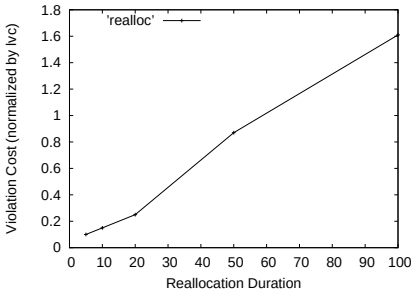

Fig. 21. Reallocation Time

actually expires.

5) Reallocation Time: Scenario VII: Two sites S_1 and S_2 each of which is of type ST_4. Site S_1 has one workload W_1 of type WT_2 and site S_2 has one workload W_2 of type WT_2. Priority of workload W_i is i ($1 \leq i \leq 2$).

In this section, we measure the effect of node reallocation duration with violation cost. If the reallocation time is very high then one would have to be very conservative about borrowing / donating nodes. We assume that reallocation is non-preemptable, that is, once a node reallocation operation has begun it cannot be aborted. Also, we assume that when a node is being reallocated, it is unusable. Figure 21 shows the violation cost as the amount of time taken to perform a reallocation increases. Reallocation is shown as a percentage of the mean burst time. Note that the values shown in the figure have been normalized by 'lvc', the violation cost when sites are non-cooperating. When the reallocation time is very high the violation cost for cooperating multi-sites becomes larger than that of non-cooperating sites (normalized value > 1). When the lease time exceeds about 50% of the mean burst time, the violation costs of cooperating multi-sites surpasses that of non-cooperating sites.

VII. Validation

In this section, we present a validation of our analytical model against a real implementation. We present two scenarios A and B. Scenario A is used to quantitatively compares the results obtained from our policy evaluation tool to a real implementation. Scenario B illustrates how we could use our policy evaluation tool to compare and refine policies.

A. Scenario A

Scenario A used for experimental measurement is as follows: We have two sites S_1 and S_2. Site S_1 has three nodes s_{11}, s_{12} and s_{13}; and site S_2 has one node s_{21}. All the nodes in sites S_1 and S_2 belong to the same pool-type: Intel Pentium III 1 GHz processors with 256 MB RAM running RedHat Linux release 7.1. Site S_1 hosts the trade2 application (trade2 is J2EE application publicly available from IBM [8]); and site S_2 has no workloads. Site S_1 runs the workload manager [35] for trade2 and the site resource manager (TIO [5]); site S_2 runs only its site resource manager (TIO [5]). We use a client-side HTTP load generator program that dynamically adjusts the load according to the workload model. More specifically, we used workload of type WT_1 for our experimental run. The workload manager uses the Network Dispatcher [3] to distribute this load among all the nodes assigned to workload trade2. We use a simple registry wherein all site resource managers register themselves upon startup so that the resource managers can identify their counterparts. All interactions between the resource managers are implemented as web services [39] that use Simple Object Access Protocol (SOAP [38]) messages on HTTP.

Figures 22 validates our analytical model against our prototype by comparing estimated costs from our model and the measurements obtained from our prototype. Costs compared are violation cost 'vc', remote node cost 'rsc' and reallocation cost 'rc'. Figure 22 validates the numerical solution obtained for our analytical model when the workload events are exponentially distributed. Figure 23 validates the simulation results when the workload events are Pareto distributed. We ran this experiment for 8000 seconds (little over 2 hours). From the experimental run we measured the workload violation cost (VC), remote node cost (RSC) and the reallocation cost (RC). These costs are expressed as the fraction of time (over this duration of 8000s) wherein the workload's SLA was not met (VC), a remote node was used (RSC) and the remote node was being reallocated (RC). Figure 22 shows the results obtained from our analytical model and the real measurements, when the workload events follow an exponential distribution. Figure 23 shows the results obtained from our simulation model and the real measurements, when the workload events follow a heavy-tailed Pareto distribution.

B. Scenario B

Scenario B used in our experimental evaluation shows how our policy evaluation tool can be used to compare one or more policies and refine them. In this scenario we have two sites S_1 and S_2 each have 3 nodes. Each site has the same set of 3 workloads W_1, W_2 and W_3 with priority of workload W_i being i. For each of the workloads we used the Trade2 application. The system has two policy sets PS_1 and PS_2. Policy set PS1 has the following policies: thr_br = 1, thr_dn = 90%, $lease\ time$ = 600 seconds, preempt remote workload = true. We use $thr\_dn = x$ to denote that a site donates nodes to remote workloads only if it its average node utilization is lower than x%. When preempt remote workload is true, the site can prematurely break a lease on node that it has donated to a remote workload. Policy set PS_2 has the following policies: thr_br = 2, thr_dn = 80%, $lease\ time$ = 600 seconds, preempt remote workload = false. When preempt remote workload is false, a node leased to a remote workload cannot be prematurely broken. However, a request to extend a lease (for the next 600 seconds) can be declined.

Figure 24 shows the total cost (violation + reallocation + remote node cost) for the policy sets PS_1 and PS_2 Vs the mean load. While PS_1 performs better at lower loads, at higher loads the

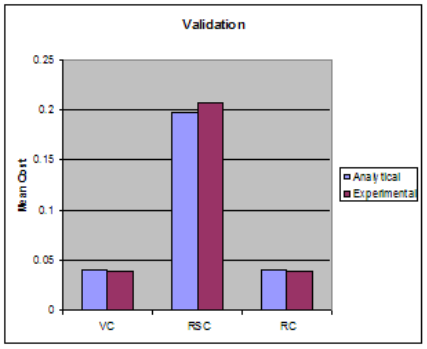

Fig. 22. Exponentially Distributed Events

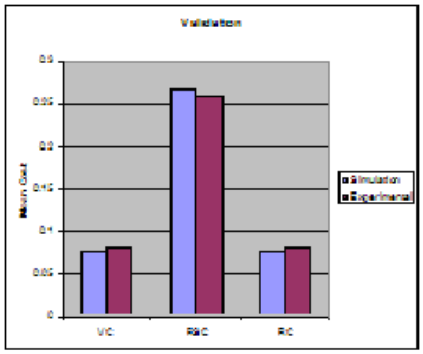

Fig. 23. Pareto Distributed Events

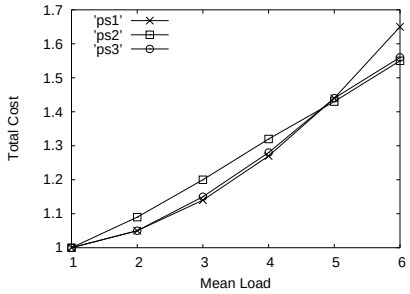

Fig. 24. Policy Refinement

policy set PS_2 performs better. This is because of the following reasons. (i) At high loads number of borrows could be high, thereby increasing the reallocation cost, (ii) The conservative nature of policy set PS_2 reduces the reallocation cost (higher borrow threshold and lower donation utilization threshold), and (iii) Further, we observed that the policy set PS_1 donates nodes too often and breaks the lease too often (at high load), thereby increasing its reallocation costs significantly.

The policy evaluation tool helps us identify the crossover point between the policy sets PS_1 and PS_2. It also helps the administrator to construct a policy set PS_3 from PS_1 and PS_2 that imitates PS_1 when the low load and imitates PS_2 at high load. Figure 24 shows the total cost incurred by policy set PS_3 from our real-implementation. Note that the costs for policy sets PS_1 and PS_2 in figure 24 are obtained from our policy evaluation tool.

In summary, our initial experiments show that our tool is capable of estimating these costs within a precision of about 5%. We found that a real experiment has to be run for at least one hour for the workload costs (per unit time) to converge to some steady value. On the other hand, our numerical solution can be computed in a couple of seconds; and our simulation model requires only a few tens of seconds before the cost values converge to a steady-state value. This makes it possible for a policy maker to interactively use this tool to define and fine-tune a site's policies.

VIII. Conclusion

This paper presents an analysis methodology for studying the effect of resource sharing policies on a multi-site resource management system. We demonstrated that our tool could be very useful for system designers in building effective solution strategies, and the methodology can be incorporated into a planning and system management tool that would permit policy-makers to understand, fine-tune and analyze the effect of policies on the system. We have presented a validation of our model against a prototype implementation of multi-site resource management system. A summary of the key results obtained by using our analysis methodology on multi-site resource management include: (i) Moderate mean-load and high load variance are best handled by cooperating sites. (ii) Borrow threshold policy is useful only under high load; at lower loads a high borrow threshold tends to increase the aggregate workload costs. (iii) Short lease times are good for high priority workloads. In conclusion, our tool is capable of evaluating complex policies on a multi-site resource management system and permits independent policies for each site, so that policy makers can quickly evaluate several alternatives and their effects on the workloads before deploying them.

As a part of our future work we are exploring three directions. First, we are exploring techniques to extend our policy analysis tool to operate with general distributions for node failure and recovery model and workload models. We plan to use G/M/1 queuing networks (using embedded Markov chains [33]) and G/G/1 queuing networks (using well-known approximations [33]). Second, we plan to explore the economics of multi-site resource allocation wherein each site is operated by competing agents using game theoretic techniques. Third, we intend applying our techniques to perform policy evaluation in other application domains, in particular, policy evaluation for SANFS (storage area network file system).

Acknowledgements. This work is motivated by a larger effort in multi-site resource allocation - we are indebted to our colleagues on this project, Asit Dan, Daniela Rosu, Vijay Naik, Sameh Fakhouri, and Daniel Dias. They have also given us specific advice that helped this work greatly. We thank Mark Squillante for reviewing the work and giving us valuable feedback. We want to thank Daniel Dias's skillful management support for this effort as well as for the multi-site resource allocation project.

References

[1] Globus toolkit. http://www.globus.org/.
[2] IBM data center solution. http://www-1.ibm.com/servers/eserver/xseries/windows/datacenter.html.
[3] IBM network dispatcher features. http://www-3.ibm.com/software/network/about/features/keyfeatures.html.
[4] IBM tivoli decision support for OS/390 capacity planner feature guide and reference. http://publib.boulder.ibm.com/tividd/td/TDS390/SH19-4021-05/en_US/HTML/drlz9mst.htm.
[5] IBM tivoli intelligent orchestrator. http://www-306.ibm.com/software/tivoli/products/intell-orch.
[6] IBM tivoli provisioning manager. http://www-306.ibm.com/software/tivoli/products/prov-mgr.
[7] IBM websphere extended deployment edition (websphere XD). http://www-6.ibm.com/jp/software/websphere/ft/was/xd/pdf/whitepaper.pdf.
[8] IBM websphere performance benchmark sample (trade 2). http://www-4.ibm.com/software/webservers/appserv/wpbs_download.html.
[9] LPAR: Dynamic logial partitioning. http://www-03.ibm.com/servers/eserver/iseries/lpar/.
[10] Platform LSF. http://www.platform.com/.
[11] VMWare ESX server. http://www.vmware.com/.
[12] WS agreement specification. http://www.gridforum.org/.
[13] E. A. Brewer. Lessons from giant scale services. In *IEEE Internet Computing*, 2001.

[14] G. Cheliotis and C. Kenyon. Autonomic economics: Why self-managed e-business systems will talk money. In *IEEE Conference on E-Commerce*, 2003.
[15] G. Cheliotisyx, C. Kenyony, and R. Buyya. 10 lessons from finance for commercial sharing of it resources. In *Peer to Peer Computing: The evolution of disruptive technology, IRM Press*, 2004.
[16] B. Cooper and H. Garcia-Molina. Peer-to-peer resource trading in a reliable distributed system. In *International Workshop on Peer-to-Peer Systems*, 2002.
[17] Y. Drougas and V. Kalogeraki. A fair resource allocation algorithm for peer-to-peer overlays. In *Global Internet*, 2005.
[18] C. Dumitrescu and I. Foster. Usage policy based cpu sharing in virtual organizations. In *5th IEEE/ACM International Workshop on Grid Computing*, 2004.
[19] C. Dumitrescu, M. Wilde, and I. Foster. A model for usage policy-based resource allocation in grids. In *IEEE Policy Workshop*, 2005.
[20] G. S. Fishman. Discrete event simulation. Spirnger Series in Operations Research, ISBN: 0387951601.
[21] I. Foster, D. Gannon, H. Kishimoto, and J. V. Reich. Open grid services architecture use cases. Information Document, Global Grid Forum (GGF).
[22] I. Foster, C. Kesselman, J. Nick, and S. Tuescke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.
[23] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. In *International Journal of Supercomputer Applications, 15(3): 200-222*, 2001.
[24] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The open grid services architecture. Informational Document, Global Grid Forum (GGF), 2005.
[25] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster based scalable network services. In *16th ACM SOSP*, 1997.
[26] H. Gimpel, H. Ludwig, A. Dan, and R. Kearney. PANDA: Specifying policies for automated negotiations of service contracts. In *International Conference on Service Oriented Computing*, 2003.
[27] Globus. Grid resource allocation management (GRAM) service. http://www.globus.org/toolkit/docs/3.2/gram/.
[28] A. K. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large scale web server access patterns and performance. In *World Wide Web Journal*, 1999.
[29] J. Kay and P. Lauder. A fair share scheduler. Univ of Sydney and AT&T Labs, 1988.
[30] K. H. Kim and R. Buyya. Policy-based resource allocation in hierarchical virtual organizations for global grids. In *18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD06)*, 2006.
[31] K. H. Kim and R. Buyya. Fair resource sharing in hierarchical virtual organizations for global grids. In *8th IEEE/ACM International Workshop on Grid Computing*, 2007.
[32] A. J. King and M. S. Squillante. Optimal control of web hosting systems under service level agreements. In *IBM Research Report RC23094 (W0401-145)*, 2004.
[33] L. Kleinrock. Queuing systems. In *Wiley Interscience ISBN 100471491101*, 1975.
[34] O. Kremien and J. Kramer. Methodical analysis of adaptive load sharing algorithms. In *IEEE transactions on parallel and distributed systems, vol 3, No 6, November 1992*.
[35] A. Leff, J. Rayfield, and D. Dias. Meeting service level agreements in a commercial grid. In *IEEE Internet Computing*, 2003.
[36] H. Ludwig, A. Keller, A. Dan, R. King, and R. Frank. A service level agreement language for dynamic electronic services. In *Electronic Commerce Research (Volume 3)*, 2003.
[37] M. S. Squillante, D. D. Yao, and L. Zhang. Web traffic modeling and server performance analysis. In *IEEE Conference on Decision and Control*, 1999.
[38] W3C. Simple object access protocol (SOAP). http://www.w3.org/TR/soap.
[39] W3C. Web services. http://www.w3.org/2002/ws.
[40] Y. Yan, A. El-Atawy, and E. Al-Shaer. Ranking-based optimal resource allocation in peer-to-peer networks. In *IEEE Infocom*, 2007.