Policy-Based Resource Assignment in Utility Computing Environments

Cipriano A Santos, Akhil Sahai, Xiaoyun Zhu, Dirk Beyer, Vijay Machiraju, Sharad Singhal

HP Laboratories, Palo-Alto, CA, USA
{psantos, asahai, xiaoyun, dbeyer, vijaym, sharad}@hpl.hp.com

Abstract. In utility computing environments, multiple users and applications are served from the same resource pool. To maintain service level objectives and maintain high levels of utilization in the resource pool, it is desirable that resources be assigned in a manner consistent with operator policies, while ensuring that shared resources (e.g., networks) within the pool do not become bottlenecks. This paper addresses how operator policies (preferences) can be included in the resource assignment problem as soft constraints. We provide the problem formulation and use two examples of soft constraints to illustrate the method. Experimental results demonstrate impact of policies on the solution.

1 Introduction

Resource assignment is the process of assigning specific resources from a resource pool to applications such that their requirements can be met. This problem is important when applications are provisioned within large resource pools (e.g. data centers). In order to automate resource assignment, it is important to convert user requests into specifications that detail the application requirements in terms of resource types (e.g. servers) and the network bandwidth required between application components. This application topology is then mapped to the physical topology of a utility computing environment. The Resource Assignment Problem (RAP) specification [1] describes this process. In RAP, applications are mapped to the topology of a utility computing environment. While RAP accounts for constraints imposed by server, storage and networking requirements during assignment, it does not consider policies that may be desirable by operators, administrators or users. In this paper we discuss how operator preferences (policies) may be incorporated as logical constraints during resource assignment. We present formulations and experimental results that deal with classes of users and resource flexing as examples of policies that may be used during resource assignment.

Policies have been traditionally considered as event-action expressions that are used to trigger control actions when certain events/conditions occur [2], [3]. These

policies have been applied in network and system management domain by triggering control actions as a result of threshold-based or time-based events. Sahai *et al*. [4] have formulated policies as hard constraints for automated resource construction. Other related work [5]-[8] on constraint satisfaction approaches to policy also treats policy as hard constraints.

In this paper, we describe policies as soft constraints for resource assignment. To the best of our knowledge, earlier work on resource assignment [1], [9] has not explored usage of soft constraints in resource-assignment. It is important to emphasize that the assignment system may violate soft constraints to varying degrees in order to ensure a technically feasible solution. In contrast, hard technological constraints, such as capacity limits, cannot be violated during resource assignment because their violation implies technological infeasibility.

The rest of this paper is organized as follows. In Section 2, we review the resource assignment problem, and present the mathematical optimization approach to resource assignment. Section 3 describes how policy can be incorporated in this problem as soft constraints. It also presents the formulation for incorporating class-of-user policies during resource assignment as well as for application flexing. Simulation results using this approach are described in Section 4. We conclude with some directions for future work in Section 5.

2 An Optimization Problem for Automated Resource Assignment

In [1], a resource assignment problem (RAP) for a large-scale computing utility, such as an Internet data center, was defined as follows. Given the topology of a physical network consisting of switches and servers with varying capabilities, and for a given component-based distributed application with requirements for processing and communication; decide which server from the physical network should be assigned to each application component, such that the traffic-weighted average inter-server distance is minimized, and the application's processing and communication requirements are satisfied without exceeding network capacity limits. This section briefly reviews the models used to represent computing resources and applications. The reader is referred to [1] for more details.

2.1 The RAP Models

Figure 1 shows an example of the physical network. The network consists of a set of switches and a set of servers connected in a tree topology. The root of the tree is a switching/routing device that connects the fabric to the Internet or other utility fabrics. All the internal nodes are switches, and all the leaf nodes are servers. Note that the notion of a "server" here is not restricted to a compute server. It includes other devices such as firewalls, load balancers, network attached storage (NAS), VPN gateways, or other such components. Each server is described by a set of attribute values, such as processor type, processor speed, memory size, etc. A complete list of parameters that characterize the network topology and resource capabilities is available in [1].

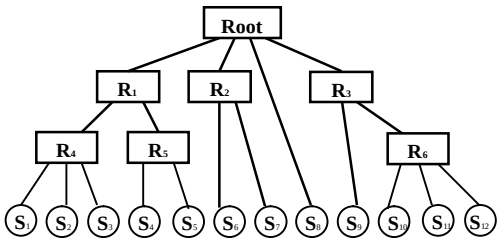

Fig. 1. Topology of a physical network

Figure 2 shows the component architecture of a distributed application, which is represented by a directed graph G(C, L). Each node $c \in C$ represents an application component, and each directed edge $l = (c, c') \in L$ is an ordered pair of component nodes, representing communication from component c to component c'. The bandwidth requirement is characterized by a traffic matrix *T*, where each element $T_{cc'}$ represents the amount of traffic from component *c* to component *c'*. Each component has a set of requirements on the attributes of the server that will be assigned to it.

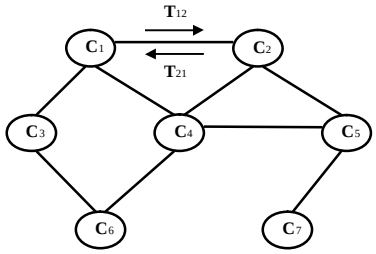

Fig. 2. A component-based distributed application architecture

2.2 A Mathematical Optimization Approach

The very large number of resources and inherent complexity of a computing utility impose the need for an automated process for dealing with RAP. Two elements make a decision problem: First there is the set of alternatives that can be followed – "like knobs that can be turned." Second, there is a description of what is "allowed", "valid", or "feasible". The task of the decision maker is to find a "setting of the knobs" that is "feasible." In many decision problems, not all feasible settings are of equal desirability. If there is a way of quantifying the desirability of a setting, one can ask to find the best of all feasible settings, which results in an optimization problem. More formally, we model the RAP optimization problem with three elements:

- *The decision variables* describe the set of choices that can be made. An assignment of values to all decision variables constitutes a candidate solution to the optimization problem. In RAP, the decision variables represent which server in the computing utility is assigned to each application component.
- *The feasible region* represents values that are allowed for the decision variables. Typically not all possible combinations of values assigned to the decision variables denote an assignment that meets all technical requirements. For example, application components may have processing or communication requirements that cannot be satisfied unless those components are assigned to specific servers. These requirements are expressed using equality or inequality constraints.
- *The objective function* is a measure of goodness of a given assignment of values to all decision variables, expressed as a parameterized function of these decision variables. In [1], we chose a specific objective function for RAP that minimizes the traffic-weighted inter-server distance. However, the formulation is flexible enough to accommodate other choices of goodness measures, such as costs, or certain utility functions.

We chose mathematical optimization as a technique to automate the resource assignment process primarily for its expressive power and efficiency in traversing a large and complex search space. Arriving at a solution that is mathematically optimal within the model specified is a welcome side effect. Therefore, RAP was formulated as a constrained optimization problem. We were not interested in developing our own optimization technology, so we chose to use off-the-shelf optimization tools. Through proper linearization of certain constraints in RAP, we derived a *mixed integer program* (MIP) [10] formulation of the problem. Our prototype solver is implemented in the GAMS language [11], which generates a MIP model that can be fed into the CPLEX solver [12]. The latter either finds an optimal/sub-optimal solution that denotes a technically feasible and desirable assignment of resources to applications, or declares the problem as infeasible, which means there is no possible assignment of resources to applications that can meet all the technical requirements.

A detailed description of the MIP formulation is presented in [1]. Note that the model in [1] also contains a storage area network (SAN) in the utility fabric and includes applications' requirements on storage. In this paper, only policies and rules that are directly related to server resources are considered. If necessary, policies for storage resources can be easily incorporated in a fashion similar to those described here.

3 Incorporating Policies in Resource Assignment

In addition to technical constraints described above, we need to include operator policies and business rules during resource assignment. For example, it may be important to consider application priority when resources are scarce, or components migration policies during application flexing.

Operator policies and business rules are often expressed as logical statements that are actually preferences. The operator would like these preferences to be true, as long as other hard constraints are not violated. The set of operator policies for an

assignment itself defines a feasible region of decision variables. Replacing the feasible region of the original problem with the intersection of that region and the feasible region defined by operator policies provides the region of all feasible assignments that meet technical requirements and operator policies at the same time. Because a wide variety of operator policies can be expressed by the decision region formed by linear inequalities, they can be incorporated into the resource assignment problem during mathematical optimization.

The concept of hard and soft constraints developed in the context of mathematical programming provides a valuable tool to handle operator policies in the context of assignment. Hard constraints are stated as inequalities in an optimization problem. Any assignment that violates any of such constraints is identified as infeasible and not a viable solution. In general, we consider that constraints imposed by the technology are hard constraints that cannot be violated (i.e., their violation implies technical infeasibility of the solution). On the other hand, constraints imposed by rules, policy, or operator preferences are soft constraints that may be violated to varying degrees if a solution is otherwise not possible. This is accomplished by introducing a variable v that measures the degree of violation of a constraint. More formally, let a policy constraint be given by

$$f(x) \leq b,$$

where x is the vector of decision variables, the function $f(x)$ encapsulates the logic of the constraint and the scalar b stands for a desirable threshold. In the above formulation, the constraint is hard. Any valid assignment x must result in a function value $f(x)$ which is not larger than b. By introducing the violation variable v in the form

$$f(x) \leq b + v,$$

we see that for any choice of x, the variable v will have to take a value $v \geq f(x) - b$ which is at least as big as the amount by which the original constraint is violated. Nonetheless, whatever the particular choice of x, the soft constraint can be satisfied. This alone would render the new constraint meaningless. In order to compel the optimization algorithm to find an assignment x that violates the constraint only as much as necessary to find an otherwise feasible solution, we introduce a penalty into the objective function that is proportionate to the violation itself by subtracting [1] the term $M \cdot v$. If M is a sufficiently large number, the search for the optimal solution will attempt to minimize the violation of the constraint and only consider a violation if there is no feasible solution that satisfies all constraints.

The typical operator/customer policies related to resource assignment in a utility computing environment that can be handled by an optimization approach include the following:

- Priority policies on classes of applications.
- Migration policies during application flexing.

[1] This assumes that our goal is maximizing the objective. If we want to minimize the objective we simply add the same term.

- Policies for avoiding hot spots inside the resource pool, such as load balancing, or assigning/migrating servers based on local thermal conditions.
- Policies for high availability, such as dictating redundant designs, or maintaining buffer capacities in shared resources.
- Policies for improving resource utilization, such as allowing overbooking of resources.

In what follows, we use the first two policies as examples to illustrate how these policies can be incorporated into the original RAP MIP formulation. The other policies can be dealt with in a similar fashion.

3.1 Policies on Classes of Applications

In a resource constrained environment it is useful to consider different classes of applications, corresponding to different levels of service, which will be reflected in terms of priorities during resource assignment. If resources are insufficient to satisfy all applications, low priority applications are more likely to be rejected when making assignment decisions. In this paper, we consider the following priority policy:

P1. *Only assign an application with lower priority to the computing utility if its assignment does not preclude the assignment of any application of higher priority* .

While this policy has a very complex logical structure, it is easy to implement by using soft constraints. Let the binary decision variable $x_{c,s} = 1$ indicate that component *c* is assigned to server *s*, otherwise $x_{c,s} = 0$. Let *C*(*app*) be the set of all components of application *app* with |*C*(*app*)| denoting the number of components of the respective application. Then the "hard constraint"

$$\sum_{s \in S} x_{c,s} = 1 \ , \quad c \in C(app) \qquad \text{(H1)}$$

implies that at an application component should be assigned to exactly one server. It can be relaxed as follows:

$$\sum_{s \in S} x_{c,s} \le 1 \ , \quad c \in C(app) . \qquad \text{(S1)}$$

The constraint (S1) means that each application component is either not assigned, or is assigned to at most one server. To disallow partial assignment (where only some of the application components are assigned) the following hard constrain t is used:

$$\sum_{c \in C(app)} \sum_{s \in S} x_{c,s} = \left| C(app) \right| . \qquad \text{(H2)}$$

It simply says that the number of servers assigned to an application is equal to the number of components required by the application. Now we introducing a binary violation variable v_{app} to relax the hard constraint (H2) as follows,

$$\sum_{c \in C(app)} \sum_{s \in S} x_{c,s} \ge \left| C(app) \right| * \left(1 - v_{app}\right). \qquad \text{(S2)}$$

It is easy to see from (S2) that,

$$v_{app} \geq 1 - (\sum_{c \in C(app)} \sum_{s \in S} x_{c,s}) / |C(app)|.$$

When all components of application *app* are placed on servers, $v_{app} \geq 0$. On the other hand, since v_{app} is binary, if any component of application *app* does not get a server, in which case application *app* has to be rejected, $v_{app} = 1$. If the term $M_{App} v_{App}$ is added onto the objective function, not assigning an application c omes at a price of M_{App}. By choosing the magnitude of M_{App} according to the application's priority in such a way that higher priority applications have penalties that are larger than all potential penalties of lower pr iority applications combined, we can force the optimal solution of the modified assignment problem to conform to priority policy P1.

3.2 Migration Policies for Application Flexing

We use the term "application flexing" to refer to the process of adding addi tional resources to or removing resources from running applications. In this section we consider policies that are related to flexing applications. Of particular interest are policies dictating whether or not a component of an application can be migrated t o accommodate chan ging resource requirements of the applications in the environment. Let C^{placed} be the set of components of running applications that have been placed on servers of the computing utility. Every co mponent is currently placed on one server. This assignment can be expressed as a comp onent-server pair. Let *ASSIGN* be the set of existing assignments, i.e.,

$$ASSIGN = \{(c, s) : \text{component } c \text{ is assigned to server } s\}.$$

We denote the set of components that can be migrated as $C^{mig} \subseteq C^{placed}$ and the set of components that cannot be migrated as $C^{nomig} = C^{placed} - C^{mig}$. Let us consider the following migration policy:

P2. *If an application component is not migratable, it should remain on the server it was placed on; if a component is migratable, migration should be a voided unless feasible assignments meeting new application requirements can not be found otherwise.*

Prohibiting migration of the components in C^{nomig} is accomplished by introducing the following additional constraints: For each assig nment, $(c, s) \in ASSIGN$,

$$x_{c,s} = 1 \quad c \in C^{nomig}.$$

For components that can be migrated, P1 states that migration should be avoided unless necessary. This is incorporated by introducing a penalty p^{mig} in the objective function for changing the assignment of an existing component. Thus, we add

$$\sum_{\substack{(c,s)\in ASSIGN \\ c\in C^{mig}}} p^{mig}(1-x_{c,s})$$

to the objective function. It is easy to see that the penalty is incurred whenever a component is moved away from its current server, i.e. when $x_{c,s} = 0$.

4 Simulation Results

In this section, we present simulation results of two resource assignment scenarios that required the two policies described in Section 3, respectively. The first simulation shows the use of priorities in assigning resources to applicati ons when the available resources are insufficient to meet the demands of all applications. The second simulation demonstrates the impact of policies around mobility of application components in an application flexing scenario.

4.1 Description of the Computing Utility

The computing utility considered in our simulations is based on a 125 -server utility data center [1] in HP Labs. The physical network has two layers of switches below the root switch. We refer to the one switch that is directly connected to the root switch as the edge switch (*e1*), and the four additional switches that are directly connected to the edge switch as the rack switches (*r1-r4*)[2]. There are no direct connections between the rack switches. All the 125 servers are either connected to the edge switch, or to a rack switch. Table 1 describes the exact network topology of the utility.

Table 1. Network topology of the computing utility

Type of switch	Edge	Rack			
Switch label	*e*1	*r*1	*r*2	*r*3	*r*4
No. of directly-connected servers	61	12	12	20	20

Among the 61 servers directly-connected to *e*1, there are 15 high-end servers in terms of their processing capacity. All the switches in the utility are non -blocking. As a result, if all traffic of an application is contained within one switch, network bandwidth is not an issue. If traffic has to traverse switches, inter -switch link capacity, as we will see, can become a scarce resource.

[2] Each switch has a hot standby for high availability. However, in the logical topology of the network, only the primary switch is considered.

4.2 Description of the Applications

In both simulations, we consider 10 applications that need to be hosted in the computing utility. The application topology considered is a three-tier topology typical of e-commerce applications. The resource requirements of the applications follow:

1. Application components do not share servers. Thus every application requires a separate server for each of its components.
2. Each application contains a high-end component for its back-end component (typically a database). Thus each application requires one high-end server.
3. The total amount of network bandwidth needed by each application can be classified into three categories: High, Medium, and Low.
4. Based on the criticality of meeting the application's resource demand, each application belongs to one of the three priority classes: Platinum, Gold, and Silver.

Table 2. Resource requiredments of the 10 applications

Application number	1	2	3	4	5	6	7	8	9	10
Total no. of components	8	8	10	5	7	8	6	10	5	6
High-end components	1	1	1	1	1	1	1	1	1	1
Bandwidth requirements. (Hi / Med / Low)	H	M	H	M	M	M	L	H	L	M
Application priority (Platinum/Gold/Silver)	P	P	G	P	P	P	S	P	S	P

These requirements are summarized in Table 2. Notice that, since a total of 73 servers are needed, not all applications can fit simultaneously on the 61 servers directly connected to $e1$. As a result, some applications will have to be allocated in a way that traffic traverses switches creating potential network bottlenecks.

4.3 Policies on Classes of Applications

In the first simulation, we consider the problem of assigning resources to the 10 applications simultaneously. We compare two approaches for undertaking the assignment: without any priority policies or with the priority policy P1 defined in Section 3.1. The result of the comparison is illustrated in Fig. 3.

As we can see, when no priority policies are implemented, all the applications are assigned resources from the computing utility except *App*8 – a platinum application. This result is intuitive, because when priority levels of applications are ignored, the RAP solver first tries to place the largest number of applications possible, second it chooses those applications that minimize the traffic weighted inter-server distance as described earlier. In our scenario, this results in excluding placement of *App*8 since it requires a large number of servers and high bandwidth.

As explained in Section 3.1, when application priorities are enforced, the priority policy P1 is incorporated into the RAP optimization problem using soft constraints, i.e., adding a penalty onto the objective function when the policy is violated. As

indicated by the third column in Fig. 3, the resulting assignment solution is different. Now *App*3 in the "Gold" class is not assigned while *App8* in the "Platinum" class is. This simulation demonstrates the impact of including the priority policy on the resulting assignment solution. It also validates the value of the soft constraint approach for incorporating priority policies into our RAP solver.

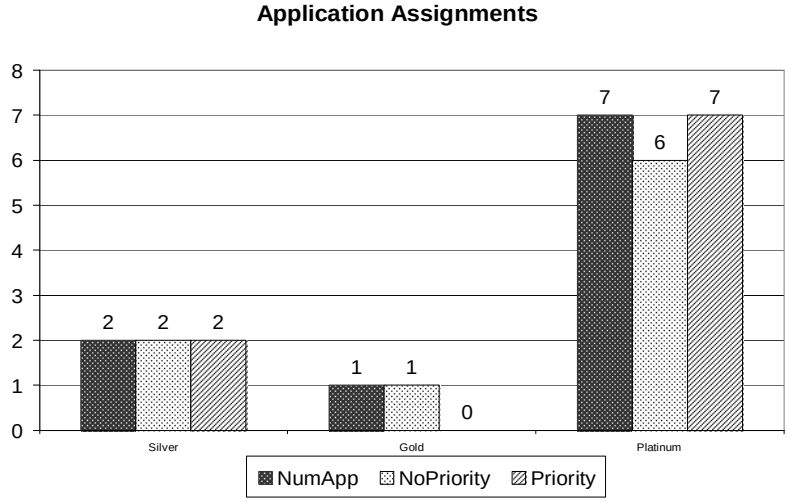

Fig. 3. Total number of applications, number of applications placed without priority, and number of applications with priority policy P1 in each priority class

4.4 Migration Policies for Flexing Applications

In this simulation, we consider an application flexing scenario and demonstrate the impact of the migration policy P2 we defined in Section 3.2. Consider the assignment obtained using the priority policy in the last section. For this assignment, all servers directly connected to the switch *e*1 are assigned to applications, including the 15 high-end servers. However, the hosted nine applications together require only nine high-end servers. As a result, six high-end servers are used to host components that could have been hosted on a low-end server, and therefore, no high-end servers are currently available for assignment.

Let us now assume that after a while of running the nine applications in the computing utility, some applications' resource demands change: *App*8 is requesting one additional high-end server, while *App*10 is able to release three low-end components that happen to be placed on low-end servers[3]. It is obvious that if no migration of application components is permissible, *App8*'s flexing request cannot be

[3] The traffic requirements of the flexed applications have been adjusted accordingly in the input data. Since both applications only use servers directly connected to the edge switch *e*1, the traffic of these two applications is not affecting the assignments described below.

satisfied, because even after the release of the servers no longer needed by *App*10 there are no more high-end servers available in the free pool.

On the other hand, treating all components as migratable and, in essence, solving a new initial assignment problem for all nine applications currently admitted may prescribe a new assignment that requires moving many components resulting in severe disruption of service for many of the applications.

The solution lies in specifying sensible migration policies that can be taken into account by the RAP solver. Let's consider the following migration policy on top of the formerly defined policy P2: existing low-end components can be migrated, while existing high-end components have to stay put. This is reasonable because for example, for a 3-tier Web application, the low-end components are Web servers and application servers that are more likely to be migratable, while high-end components can be database servers that are much harder to move.

As described in Section 3.2, the above migration policy was implemented by adding both hard and soft constraints to the RAP MIP formulation. Table 3 shows the resulting assignment of high-end and low-end servers to applications before and after flexing. Only the applications affected by flexing are shown. All the other assignments remain the same. As we can see, by incorporating the above migration policy, the RAP solver finds an assignment for the flexed applications, where one low-end component of *App1* previously assigned to a high-end server is migrated to a low-end server released by *App*10, and this freed high-end server is used to host the additional high-end component of *App*8.

Table 3. Server assignment to the applications affected by flexing

Applications		*App*1		*App*8		*App*10	
Servers		Low-end	Hi-end	Low-end	Hi-end	Low-end	Hi-end
Before flexing	Required	8	1	10	1	6	1
	Assigned	7	2	10	1	6	1
After flexing	Required	8	1	10	2	6	1
	Assigned	8	1	10	2	3	1

This simulation demonstrates that, by defining sensible migration policies based on properties of application components and server technologies, we are able to accommodate flexing requests that may otherwise be infeasible, thus increasing resource utilization. At the same time, we minimize the disruption to applications that are already running in the computing utility. In addition, the result verifies that using a combination of hard and soft constraints in the optimization problem can be an effective way of incorporating migration policies into the RAP optimization problem.

5 Conclusion and Future Work

In this paper, we demonstrate how operator policies can be included in a automated resource assignment using mathematical optimization techniques. Mathematical optimization is used because, as shown in [1], a simple heuristic leads to poor

application placements that can create fragmented computing resources and network bottlenecks. Our simulation results on two resource assignment scenarios with common policies encountered in a utility computing environment confirm that our framework can not only address the resource assignment problem efficiently, but also offers a unified approach to tackle quantitative and rule based problems.

As a final note, observe that policies and rules need to be defined precisely in a way that helps to answer the quintessential question for resource assignment: Can resource *s* be assigned to component *c*? Consequently, we require a data model for the business rules and operator policies that allows expressing these rules and policies in terms of the parameters and decision variables of the MIP formulation of the resource assignment problem. In the future, we may develop a tool that directly writes mathematical programming code, without the need of templates and associated data models as shown in the examples of Section 3 and 4.

References

1. X. Zhu, C. Santos, J. Ward, D. Beyer and S. Singhal, "Resource assignment for large scale computing utilities using mathematical programming," *HP Labs Technical Report, HPL-2003-243*, November 2003. http://www.hpl.hp.com/techreports/2003/HPL-2003-243R1.html
2. N. Damianou, N. Dulay, E. Lupu, Morris Sloman, "The Ponder policy specification language," *Proceedings of IEEE/IFIP Policy 2001*, p18-38.
3. PARLAY Policy Management, http://www.parlay.org/specs
4. A. Sahai, S. Singhal, R. Joshi, V. Machiraju, "Automated policy-based resource construction in utility computing environments," *HPL-2003-176, Proceedings of IEEE/IFIP NOMS 2004*.
5. A. Sahai, S. Singhal, R. Joshi, V. Machiraju, "Automated resource configuration generation using policies," *Proceedings of IEEE/IFIP Policy 2004*.
6. P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, The MIT Press, Cambridge, Mass, 1989.
7. R. Raman, M. Livny, M. Solomon, "MatchMaking: Distributed Resource Management for High Throughput Computing," *Proceedings of HPDC 98*.
8. Object Constraint Language (OCL), http://www-3.ibm.com/software/awdtools/library/standards/ocl.html#more
9. D. Menasce, V. Almeida, R. Riedi, R. Flavia, R. Fonseca and W. Meira Jr., "In Search of Invariants for E-Business Workloads," *Proceedings of the 2nd ACM Conference on Electronic Commerce, Minneapolis*, Oct. 2000, pp. 56-65.
10. L.A. Wolsey, *Integer Programming*, Wiley, 1998.
11. GAMS, www.gams.com
12. CPLEX, www.ilog.com