Optimal Resource Assignment in Internet Data Centers

Xiaoyun Zhu Sharad Singhal
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94304, USA
{xiaoyun, sharad}@hpl.hp.com

Abstract

This paper presents a resource model that describes the computing and networking components in a resource sharing environment in an Internet Data Center (IDC), a model for multi-tier applications that need to be deployed in this environment, and an optimization problem for resource assignment aimed at minimizing the communication delay between the servers. A number of techniques are introduced to search for the optimal/suboptimal solutions. These include projection of the solution set, partition of the network and pruning of the search space, and local clustering for large problems. Results are given for two examples to demonstrate the effectiveness of the algorithms.

1. Introduction

The rise of Applications Service Providers (ASPs) can be attributed to the ever-growing cost of information technology and the increasing complexity of managing the mission-critical enterprise and Internet applications. By outsourcing these applications to ASPs, large corporations and dot-coms receive the benefit of organizational efficiency, shorter time-to-revenue, as well as improved customer satisfaction. However, the level of service provided cannot be guaranteed without the support of a highly available, scalable, flexible, and secure infrastructure. Internet data centers (IDCs) aim to provide such an infrastructure to ASPs and web site hosters for planning, deploying and managing complex applications. While some data center providers only offer co-location services (e.g., leasing secure data center space and network connectivity) to third-party ASPs, others own the servers (compute and storage nodes) in the data center and provide value-added management services including security, performance monitoring, content distribution, and capacity planning. Many data center providers are themselves ASPs who manage not only the infrastructure but also the applications.

Internet data centers have grown rapidly over the past few years, both in the number of data centers built globally and in the size of each individual data center. Current data centers can contain tens of thousands of servers with high-speed network connections for both inter- and intra- data center communications. Managing both the infrastructure and the applications in such a large environment raises many challenging questions that do not exist in smaller environments.

In [7], the operational scalability for large data centers is addressed. On one hand, it is desirable to share data center resources among different customers and applications to maximize resource utilization. On the other hand, customers prefer dedicated resources for their applications that offer isolation and security as well as flexibility in the types of applications hosted. In [7] the notion of "*virtual application environments"* (VAEs) was proposed to deal with this issue. A VAE is a collection of data center resources allocated to a customer application matching the specific configuration requirements of the application. Although some resources, e.g., the network, may be shared among applications, the resources are partitioned and encapsulated such that the VAE logically appears as if it were a dedicated environment. This approach is adopted is our paper. We focus on this shared environment for resource allocation, which is drastically different from the caged environment currently available in many data centers.

To reduce time-to-market for the customers, customer applications need to be deployed within the data center in a timely fashion. In addition, the dynamic nature and high variability of the workload in many applications, especially e-business applications [6], requires that the resources allocated to each application be easily adjustable to maintain *service level agreements* (SLAs). Therefore, resource allocation and deployment of applications need to be programmable and highly automated. Moreover, because multiple customers co-exist in the data center, it is important to allocate resources intelligently to avoid bottlenecks within the data center. In this paper, we address the problem of automatically assigning resources to applications. Simply put, given a free pool of data center resources (computing, storage, networking etc.), the problem is to decide which ones to allocate to a given customer application. Certain

Proceedings of the 9th Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)
1526-7639/01 $10.00 © 2001 IEEE

constraints and cost function are posed so that resources are assigned in an optimum way instead of being randomly chosen from the free pool.

Resource assignment is needed not only when an application is initially deployed in the data center, but also when incremental assignment is invoked by the dynamic resource management service (see [7]). The latter is referred to as "*capacity on demand*," which means servers can be added to or removed from an application based on real-time workload and performance measurements. We will focus on the initial assignment problem in this paper. The algorithms presented can be easily modified to also determine which resources should be added to or removed from an application when capacity demand has changed.

1.1. Related Work

The techniques for resource management and task scheduling in parallel and distributed computing systems can be classified into two categories: dynamic scheduling and static scheduling. Dynamic scheduling deals with a continuous stream of real-time computing jobs and balances the load across available resources for better throughput, for example, scheduling of independent jobs on parallel supercomputers [2]. In contrast, static scheduling handles a set of tasks that communicate with one another, and the scheduling is done before run time. An early work by Stone [8] solved a two-processor scheduling problem using network flow algorithms. Since then, many heuristics have been proposed for similar problems [5]. However, they mostly aim at minimizing the completion time of all the tasks. The resource assignment problem in our paper falls into the category of static scheduling, but with a different objective function. This is in some sense similar to [9], which assigns a task consisting of communicating modules to a hypercube multi-computer to minimize the total communication traffic. The difference is, in [9] the compute nodes in the hypercube are homogeneous and there are no bandwidth limits on the communication links. Our work also differs from the more refined resource management for a single server discussed in [1]. While resources can be dynamically allocated with workload, the effort required to provision an application on a computer is large enough that fine-grained optimization is not possible.

The main challenge for resource management in a shared data center environment is scalability. A remotely related research effort is the *Global Grid Forum* [3], a community-initiated forum of individual researchers and practitioners focusing on the promotion and development of wide-area distributed computing technologies and applications. The commonality between the "grid" technologies and technologies for large data centers is the emphasis on dealing with large-scale systems and creating a scalable infrastructure.

Resource allocation in large-scale e-service systems has also been presented in [4], which describes optimal partitioning of services among servers using a model called "*systems of servers*". It uses high-level abstractions of both servers and services to tackle the large solution space, and employs genetic algorithms to search for sub-optimal solutions to the problem. The advantage of using an abstract system representation is the potential to apply the technique to a general class of large distributed systems. The disadvantage is its requirement on detailed modeling to guarantee crucial system and application specific information does not get lost in the abstraction.

We next define the data center resource model in Section 2. Section 3 describes a general model for typical multi-tier applications in a data center. The optimization problem is formulated in Section 4. Section 5 describes the computational complexity associated with solutions to this problem and discusses a variety of techniques that are used to reduce the complexity. We also present a *layered partition and pruning* algorithm, which allows us to find the optimum solution for medium sized problems. Section 6 gives results for two examples. Section 7 offers concluding remarks and a description of future work.

2. The Resource Model

An approach for handling manageability and scalability in large data centers is to partition its resources into smaller units that are easily replicable. We call each of these units a "*service core*". Each service core is assumed to have between 100 ~ 1000 servers and can host many customers. The resources in a service core include servers (compute and storage nodes) and networking components (switches, routers, firewalls, and load balancers). Through the use of virtual LAN technology, it is possible to partition the service core into logically independent customer topologies [7]. When planning for multiple applications in a single service core, we can either consider all applications at the same time and try to achieve the best overall resource allocation, or we can sequentially assign one application at a time, which means after one application has been planned, resources consumed by it are subtracted from the available resource pool before the next application is planned. We take the second approach because of its simplicity and its recursive nature. Moreover, this approach is closer to current practice, wherein applications are typically added or removed from data centers over time.

Resources in a service core can be classified based on their basic functionality. In this paper we consider a resource model that contains two types of resources: servers and switches, where a server can be a compute node, a storage node, or a combination of both. This section describes their physical topology and performance attributes that are important for the applications.

2.1. Notation

For any matrix X, X' is the transpose of X, $Tr(X)$ is the trace of X, and $diag(X)$ denotes the vector that appears on the diagonal of matrix X. X_{ij} represents the element of X on the ith row and jth column, while X_j denotes the jth column vector in X. Let $1_{d\times m}$ be the all-*1* matrix with dimension $d \times m$. 1_d denotes the d-dimensional column vector with all *1*'s, and $1_d'$ is the corresponding row vector. *0* is used to represent a zero scalar, a zero vector or a zero matrix, depending on the context. The operator $\otimes$ is used for the element-wise matrix multiplication. Any ordering (=,<,>) between two vectors or matrices is on an element-by-element basis. For a finite set A, $|A|$ denotes the cardinality (or size) of the set.

2.2. Topology

The overall topology of a service core is shown in Figure 1. A major concern in designing the network topology for the service core is its scalability. One implication is that a hierarchical structure is more desirable than a flat one. Our network model for a service core uses a tree-like structure with three layers of switches: a switch mesh (***SM***), a number of edge switches (***SE***) and more rack switches (***SR***). (This can be generalized to any number of layers.) The delay inside the switch mesh is ignored so that the switches that make up the mesh can be viewed as one single node in the tree. The servers (***N***) are connected to rack switches. In Figure 1, the nodes represent the servers and switches, and the edges represent the links that connect them. All the links are duplex links and traffic can go in either direction. We assume that the resources in one service core are sufficient for the application we need to deploy. So only a single service core is considered, which simplifies the problem and make the model more mathematically tractable.

The topology of the service core can be captured using three adjacency matrices: H^{ER}, H^{RN} and H^{EN}, which characterize the connectivity between ***SE*** and ***SR***, ***SR*** and ***N***, ***SE*** and ***N***, respectively. All the three matrices have the following structure, with different dimensions:

$$H = \begin{bmatrix} 1 & \cdots & 1 & & & & & \\ & & & 1 & \cdots & 1 & & \\ & & & & & & \ddots & \\ & & & & & & 1 & \cdots & 1 \end{bmatrix}.$$

For example, $H^{ER}_{pq} = 1$ means the pth edge switch is connected to the qth rack switch. It is easy to see the connection between these matrices: $H^{EN} = H^{ER} H^{RN}$.

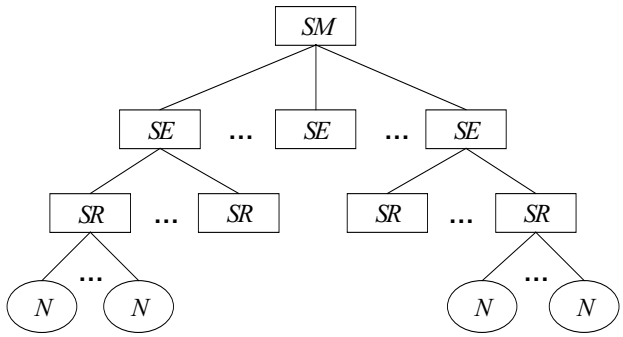

Figure 1. Topology of a service core

Three parameters are used to describe the size of the service core: N^E (no. of edge switches), N^R (no. of rack switches) and M (no. of servers). This representation does not impose symmetry on the structure of the network, although a real service core typically has a great deal of symmetry in the initial structure so that it is easier to build. The symmetry gradually diminishes as more and more resources are assigned to customer applications.

2.3. Attributes

The capacity and computing power of each server can be modeled as a set of attributes. Commonly used server attributes include processor speed, number of processors, disk capacity, disk bandwidth and memory size. There may be other attributes depending on applications. Suppose there are a total of K server attributes. The attributes can then be represented by a $K \times M$ matrix A, where A_{kj} is the kth attribute of the jth server.

Now consider the bandwidth attributes for all the links that connect the servers and the switches. Due to the hierarchical structure of the network the links have three layers. For each duplex link we use two parameters to characterize the bandwidth limit for the incoming and outgoing traffic. Here "incoming" means going down the hierarchy, and "outgoing" means going up the hierarchy. Therefore, a total of six parameters, B^{EI}, B^{EO}, B^{RI}, B^{RO}, B^{NI} and B^{NO}, are used to describe the incoming and outgoing bandwidth at the edge switches, the rack switches, and the server nodes, respectively.

3. The Application Model

In this section we describe a multi-tier model for the application that needs to be deployed in the service core.

3.1. Configuration

A typical Web application has a three-tier structure, containing front-end web servers, application servers and

back-end database servers. This concept is adopted in our application model, with every two neighboring tiers connected through a virtual LAN (VLAN). Figure 2 shows the configuration of a three-tier application. To be flexible, we assume there are D tiers in the configuration. Tier 0 is an abstraction of the connection to the Internet. T_0 can be a router, whose detail is neglected in our model. A D-dim vector C determines the distribution of servers among tiers, where C_i are assumed known *a priori*.

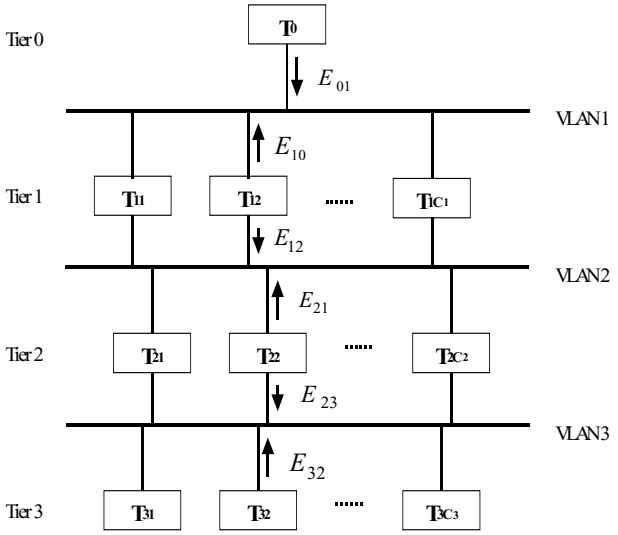

Figure 2. Configuration of an application

3.2. Requirements

Each application to be deployed has certain requirements on the attributes of the servers and the bandwidth of the network. It is conceivable that the requirements for different servers and different links in the network can be different. Again we make the following assumptions to simply these requirements.

For server attributes:

- Servers in the same tier have similar functionality. Therefore, they have uniform attribute requirements.

We assume the requirements for the server attributes can be characterized using two $D \times K$ matrices L and U, where L_{ik} and U_{ik} are the lower bound and the upper bound for the kth attribute of all the servers in the ith tier. For instance, if the servers on the first tier need to have 5~8 processors of at least 400 MHz speed, then $L_{11} = 5$, $U_{11} = 8$, $L_{12} = 400$, $U_{12} = \infty$.

For link bandwidth:

- The amount of traffic generated by different servers in the same tier is compatible, therefore is considered identical in this model.
- Traffic coming into each tier is evenly distributed among all the servers.
- No traffic goes between servers in the same tier.

We define a $D \times D$ matrix E to be the *traffic matrix* for the application, where $E_{ii'}$ indicates the maximum amount of traffic going from each server in the ith tier to each server in the i'th tier. In addition, two scalar parameters E_{01} and E_{10} are used to capture the traffic coming into and going out of the service core. Using these parameters, we can calculate the total amount of incoming and outgoing traffic at each server in different tiers, denoted by two $D \times 1$ vectors E^I and E^O, respectively. In particular, $E^O = EC + [E_{10} \quad 0 \quad \cdots \quad 0]'$, and $E^I = E'C + [E_{01} \quad 0 \quad \cdots \quad 0]'$.

Modeling and prediction of bandwidth requirements for real applications is a hard problem and is extensively studied by many researchers. In this paper we focus on the resource assignment problem, therefore, we assume these requirements are given and can be simplified and converted into the above form. The above assumptions on link bandwidth may not hold for all applications, but they are good approximations for generic multi-tier Web applications that we consider here. We will talk about relaxations to these assumptions at the end of the paper.

4. The Optimization Problem

Now we are ready to define the optimization problem for resource assignment. On one hand, the servers assigned to an application should meet the attribute requirements of the application, and the amount of traffic the application generates should not exceed the corresponding bandwidth on individual links. On the other hand, the objective of the optimization depends on the needs of each application. For example, from a data center management point of view, physical locality of the servers for a single application may be preferred, while for application performance, it is desirable to minimize the average communication delay inside the service core to improve the response time. The latter is adopted in this paper, that is, to minimize the total amount of network traffic between the servers weighted by the lengths of individual communication paths. The objective function is similar to the notion of "*communication traffic*" used in [9]. The details are discussed further in Section 4.3.

4.1. Decision variable

We need to determine which server node should be assigned to which tier. This can be represented by a $D \times M$ matrix variable X, where

$$X_{ij} = \begin{cases} 1, & \text{jth server node assigned to the ith tier;} \\ 0, & \text{otherwise.} \end{cases}$$

4.2. Constraints

The following constraints are posed on the decision variable. Due to limited space we only present the final matrix representation of the constraints. Most of the derivation that is omitted is fairly straightforward.

- The number of servers allocated to the *i*th tier is C_i.
$$X1_M = C. \qquad (1)$$
- Each server can only be assigned at most once.
$$X'1_D \le 1_M. \qquad (2)$$
- Attribute values for each server assigned satisfy the upper and lower bound conditions.
$$L'X \le (1_{K\times D} X) \otimes A \le U'X. \qquad (3)$$
- The bandwidth constraints for all the links that connect the servers to the rack switches are
$$X'E^O \le B^{NO}, \qquad 4)$$
$$X'E^I \le B^{NI}. \qquad 5)$$
- Now let's consider the bandwidth constraints for the links that connect the rack switches to the edge switches. The outgoing traffic at the *q*th rack switch should be the total amount of traffic generated by all the connected servers under this switch reduced by the traffic sent directly to the same group of servers. And it is subject to the bandwidth limit of the corresponding outgoing link. Therefore, we get
$$H^{RN} X'E^O - diag(H^{RN} X'EXH^{RN}{}') \le B^{RO}. \quad (6)$$
Similarly, the bandwidth constraints for the incoming links at the rack switches are
$$H^{RN} X'E^I - diag(H^{RN} X'EXH^{RN}{}') \le B^{RI}. \quad (7)$$
- With a similar derivation, we can get the bandwidth constraints for all the outgoing and incoming links that connect the edge switches to the mesh switches:
$$H^{EN} X'E^O - diag(H^{EN} X'EXH^{EN}{}') \le B^{EO}, \quad (8)$$
$$H^{EN} X'E^I - diag(H^{EN} X'EXH^{EN}{}') \le B^{EI}. \quad (9)$$

4.3. Objective Function

The goal of the optimization is to minimize the average communication delay inside the service core for each application without violating the above constraints. The number of hops (N_h) for each data packet to go through is used as an estimate of the communication time. For the traffic that only goes through a rack switch (F^R), $N_h^R = 2$; if it has to go through an edge switch but not a mesh switch (F^E), $N_h^E = 4$; if it has to go through the mesh switch (F^M), $N_h^M = 6$. It is desirable to give preference to the server pairs that communicate more frequently. Hence, we define the objective function as the total amount of traffic going through all the switches weighted by the corresponding number of hops, i.e.,

$$\hat{J} = N_h^R F^R + N_h^E F^E + N_h^M F^M. \qquad (*)$$

To simplify the notation, let $Y = X'EX$, $Y^R = H^{RN} Y H^{RN}{}'$, and $Y^E = H^{EN} Y H^{EN}{}'$. Y indicates the amount of traffic going between all the server pairs, while Y^R (Y^E) indicates the amount of traffic going between all the rack (edge) switch pairs. By simple calculation, we have $\hat{J} = 6C'E^O - 2Tr(Y^R) - 2Tr(Y^E)$, where $C'E^O$ is the maximum amount of overall traffic generated by all the servers. Since $C'E^O$ is a constant, instead of minimizing $\hat{J}$, we can maximize *J*, where

$$J = Tr(Y^R) + Tr(Y^E) = Tr(H^{RN} X'EXH^{RN}{}') + Tr(H^{EN} X'EXH^{EN}{}'). \qquad (**)$$

In summary, the optimization problem we need to solve is the following:

$$\max_X \quad Tr(H^{RN} X'EXH^{RN}{}') + Tr(H^{EN} X'EXH^{EN}{}')$$

$$\text{s.t.} \quad X1_M = C, \qquad (1)$$
$$X'1_D \le 1_M, \qquad (2)$$
$$L'X \le (1_{K\times D} X) \otimes A \le U'X, \qquad (3)$$
$$X'E^O \le B^{NO}, \qquad (4)$$
$$X'E^I \le B^{NI}, \qquad (5)$$
$$H^{RN} X'E^O - diag(H^{RN} X'EXH^{RN}{}') \le B^{RO}, \quad (6)$$
$$H^{RN} X'E^I - diag(H^{RN} X'EXH^{RN}{}') \le B^{RI}, \quad (7)$$
$$H^{EN} X'E^O - diag(H^{EN} X'EXH^{EN}{}') \le B^{EO}, \quad (8)$$
$$H^{EN} X'E^I - diag(H^{EN} X'EXH^{EN}{}') \le B^{EI}. \quad (9)$$

Since the objective function is quadratic and the constraints contain quadratic inequalities, the optimization involves constrained nonlinear programming, which cannot be solved by directly applying conventional linear programming packages. The binary constraint on the decision variable adds to the complexity of the problem. It is a hard problem due to its combinatoric nature. Next, we propose an algorithm that combines a number of techniques to reduce the complexity of the problem.

5. The Algorithm

In general, for a combinatorial problem, finding the global optimum is not guaranteed unless an exhaustive search method is employed. In the optimization problem defined in the previous section, the decision variable X is a matrix with $D \times M$ binary entries. A simple backtracking algorithm on the row vectors of X can be used to enumerate through all X that automatically satisfy the constraints (1) and (2). Let Ω be the set of all such X, then $|\Omega| = \frac{M!}{C_1!C_2!\cdots C_D!(M-\sum_i C_i)!}$, which goes up exponentially with M. If we refer to small, medium, and large sized problems as those that deal with a service core with M<10, 10<M<100, and M>100, respectively, then the above enumeration is only feasible for small problems. For medium and large sized problems, we need to develop more intelligent search algorithms.

5.1. Identifying infeasible servers

A typical service core environment contains servers of different types and capabilities. At the same time, different tiers in the application can have different performance requirements. Therefore, it is common that not every server is feasible for every tier. The infeasible servers can be identified based on constraints (3)-(5) and excluded from the beginning of the search. If a server class is the set of servers that have identical parameter values for the K server attributes, then there are typically a small number of server classes (W<10) in the service core. We can pre-compute an attribute feasibility matrix F^A ($D \times W$) based on constraint (3), where $F^A_{iw} = 1$ means the *wth* server class satisfies the attribute constraint of the *ith* tier, and 0 otherwise. Similarly, a bandwidth feasibility matrix F^B ($D \times M$) can be computed based on constraints (4) and (5). Let F be the combined feasibility matrix, then we only need to search X that comply with F. Let Ω_F be the set of all such X, then $|\Omega_F| << |\Omega|$. The difference depends on the number of infeasible servers for each tier.

5.2. Projection of the solution set

Another special property of the service core that we can take advantage of is the symmetry in the topology of the network, especially when each rack switch is connected to a large number of servers, and when most of the servers are still available for assignment. This property results in a great deal of redundancy in the number of possible combinations for assigning servers to different tiers. In fact, what is important is the number of servers assigned to each tier under each rack switch, which is captured in $X^R = P_{H^{RN'}}(X) = XH^{RN'}$, where $P_{H^{RN'}}$ can be considered as a *projection* map. Let Ω_F^R be the image set of Ω_F under this map. Then, the original optimization problem can be simplified and reformulated as

$$\max_{X^R \in \Omega_F^R} \quad Tr(X^{R'} E X^R) + Tr(H^{ER} X^{R'} E X^R H^{ER'})$$

$$s.t. \quad X^{R'} E^O - diag(X^{R'} E X^R) \le B^{RO}, \quad (6)'$$

$$X^{R'} E^I - diag(X^{R'} E X^R) \le B^{RI}, \quad (7)'$$

$$H^{ER} X^{R'} E^O - diag(H^{ER} X^{R'} E X^R H^{ER'}) \le B^{EO}, \quad (8)'$$

$$H^{ER} X^{R'} E^I - diag(H^{ER} X^{R'} E X^R H^{ER'}) \le B^{EI}. \quad (9)'$$

Therefore, the search algorithm can be broken into two steps. The first step solves the above optimization problem and finds one X^{R*} that maximizes J. The second step converts X^{R*} back to X^* that determines the optimal assignment for each server. This conversion is in general a one-to-many mapping. The criterion used in our algorithm is to assign more powerful, high-end servers to tiers with more stringent requirements to promote higher utilization of the servers.

As one way to compare the complexity of the reformulated optimization problem with the original one, we can compare the sizes of the two candidate solution sets. In general, $|\Omega_F^R| << |\Omega|$. How great the reduction is depends on many parameters in the problem. Figure 3 compares $|\Omega_F^R|$ with $|\Omega|$ for different values of M.

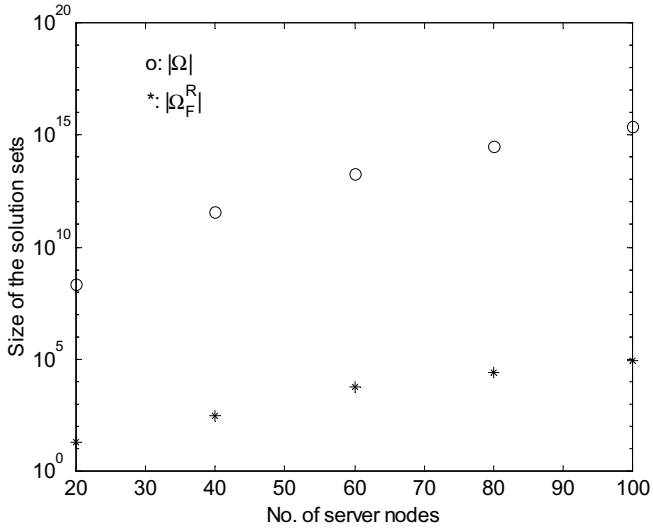

Figure 3. $|\Omega|$ vs. $|\Omega_F^R|$ for M = 20, 40, 60, 80, 100, and C = [3 4 2]

This figure clearly demonstrates the effectiveness of the above two techniques in reducing the number of candidate solutions. In this example, only the total number of servers *M* is varied while all the other parameters remain fixed. Other parameters that affect the complexity of the problem include the number of tiers in the application (*D*), the number of servers in each tier (*C*), the number of server classes (*W*), the feasibility matrix of the servers (*F*), and the mix of servers under each rack. As another example, Table 2 shows how $|\Omega|$ and $|\Omega_F^R|$ vary with *C* by fixing other parameters (*M* = 40, *D* = 3, *W* = 3). The number in the parenthesis in the first line is the total number of servers to be assigned. As we can see, although $|\Omega_F^R|$ goes up with $\sum C_i$, it grows at a much lower speed than $|\Omega|$ does. This again demonstrates how the projection technique simplifies the problem. The effect of other parameters is quite similar, hence is not shown here.

Table 1. Influence of *C* on $|\Omega|$ and $|\Omega_F^R|$

C	[3 4 2](9)	[4 5 3](12)	[5 6 4](15)
$\|\Omega\|$	3.45×10^{11}	1.55×10^{14}	2.54×10^{16}
$\|\Omega_F^R\|$	300	729	1,385

5.3. Partition of the service core

In the rest of the discussion, we focus on how to efficiently solve the reformulated optimization problem in X^R. Let X_q^R be the *q*th column vector in X^R that indicates the distribution of servers under the *q*th rack switch. Again a backtracking algorithm can be employed, which traverses the state space tree of all possible values for each X_q^R. The depth of the tree is N^R+1, and the total number of leaves in the tree equals $|\Omega_F^R|$, which is generally exponential in N^R, the number of rack switches. A standard technique for reducing the size of the state space tree in a backtracking algorithm is *pruning*. Its success relies on the ability to identify nodes that belong to infeasible or non-optimal solutions early on during the search. This is indeed possible for the above problem. In fact, constraint (6) can be rewritten as $(X_q^R)' E^O - (X_q^R)' EX_q^R \le B_q^{RO}, q=1,\cdots,N^R$, where the constraint for each X_q^R is independent. Therefore, the *q*th inequality in constraint (6) can be checked right after each X_q^R value is generated. If it is not satisfied, then the whole subtree below this particular node can be pruned. The same idea applies to constraint (7) as well. Similarly, let $X^E = XH^{EN'}$, whose each column X_p^E represents the distribution of servers under the *p*th edge switch. Then constraints (8) and (9) can be rewritten as a set of inequalities for individual edge switches that are mutually decoupled, which means each inequality can be checked for each specific X_p^E independently.

Moreover, we can rewrite the objective function as $J=\sum_{q=1}^{N^R}(X_q^R)' EX_q^R + \sum_{p=1}^{N^E}(X_p^E)' EX_p^E$. Again there is no coupling between different rack or edge switches. Let $J_p^E=(X_p^E)' EX_p^E$, and $J_p^R=\sum_{q\in Q_p}(X_q^R)' EX_q^R$, where Q_p is the index set for all the racks under the *p*th edge switch. Then $J=\sum_{p=1}^{N^E}(J_p^E+J_p^R)$. Hence, we can partition the service core by individual edge switches, and further partition the resources under each edge switch by rack switches. The resulting algorithm will be referred to as the *layered partition and pruning* (LPP) algorithm.

1. For $p=1,\cdots,N^E$, do backtracking on X_p^E, use constraints (8) and (9) to prune infeasible nodes. In the end, for each leaf X^E, compute $J^E=\sum_{p=1}^{N^E}J_p^E$.
2. For each $p=1,\cdots,N^E$, with each value of X_p^E, do backtracking on $X_q^R, q\in Q_p$ with $X_p^E=\sum_{q\in Q_p}X_q^R$. Use constraints (6) and (7) to prune infeasible nodes. Find the combination of feasible $X_q^R, q\in Q_p$ that maximizes J_p^R, record $X_q^{R*}, q\in Q_p$ and J_p^{R*}.
3. Now for each value of X^E, compute $J^R=\sum_{p=1}^{N^E}J_p^{R*}$, and $J=J^R+J^E$. Find X^{E*} that maximizes *J* and the corresponding X^{R*} recorded earlier.
4. Convert X^{R*} into X^*.

Compared to a direct backtracking algorithm on X^R that searches in Ω_F^R, the above algorithm has two main advantages. First, by partitioning the network with individual edge switches, the search of partially optimal X_q^{R*} becomes local under each edge switch. Second, by separating the edge layer from the rack layer, many evaluations only involve matrix multiplications with X_p^E,

which is of fairly low dimension. And since each X_p^E value corresponds to a set of X_q^R values, infeasible solutions are removed more quickly. All of these result in a significant reduction in the total amount of computation.

5.4. Local clustering for large problems

The above LPP algorithm is able to find the global optimal solution for medium to large sized problems with certain configuration and parameter values. When the problem becomes too large, we can aim at obtaining a suboptimal solution via a local clustering scheme. The idea is instead of searching through all the edge switches, form a group of clusters each containing a number of neighboring edge switches, pick the cluster with potentially the best solution using some heuristics, and only search inside this cluster. The reasoning behind this is the connection between server locality and reduction in the communication delay. We can imagine an assignment with all the servers located in one rack to have a lower communication delay than a distributed solution. As long as all the constraints are met, it is more desirable to have all the servers located closely. Since we use edge switches as the basis of our partition algorithm, it is natural to use them as a unit in the clustering. This approach is expected to work well for typical service cores with significant symmetry in the topology. At the early stage of application deployment, most of the servers are available, the optimization problem is large but the global optimum is likely to be found in a local cluster due to the symmetry property. As more and more servers are assigned to applications, the service core becomes less symmetric. At the same time, the optimization problem becomes smaller, when the LPP algorithm can be easily applied.

6. Case Studies

The above algorithms were tested on various instances of the problem with different parameter settings. Two examples are shown in this section. Both contain 3 classes of servers, with class 1, 2 and 3 represented by circles, diamonds and stars, respectively. The mesh, edge and rack switches are represented by squares. The lines represent the links that connect the switches and the servers, with the line width indicating the relative bandwidth. Both examples assume the application that needs to be deployed is a standard three-tier e-commerce application.

6.1. Example 1

Figure 4 shows the resulting optimal assignment in a service core with 30 server nodes (M = 30), a switch mesh, two edge switches and six rack switches. Each rack contains five servers of one class. The three rack switches under each edge switch are connected to servers of class 1, 2, and 3, respectively. The application requires 4, 5 and 3 servers for the 1st, 2nd, and 3rd tier, respectively.

Due to the small size of the service core, the LPP algorithm found the optimal solution fairly easily. Although the original solution set $|\Omega| = 2.3976 \times 10^{12}$, the reduced set Ω_F^R has only 18,774 elements after the projection. Consistent with our intuition, the optimum assignment demonstrates physical locality of the servers. As enough bandwidth is available, all the servers lie under the same edge switch so that communication between servers does not need to go through the mesh switch. In addition, since the 2nd tier needs to talk to both the 1st and the 3rd tier, the 2nd tier servers are arranged to be close to the servers on the other two tiers in a balanced way.

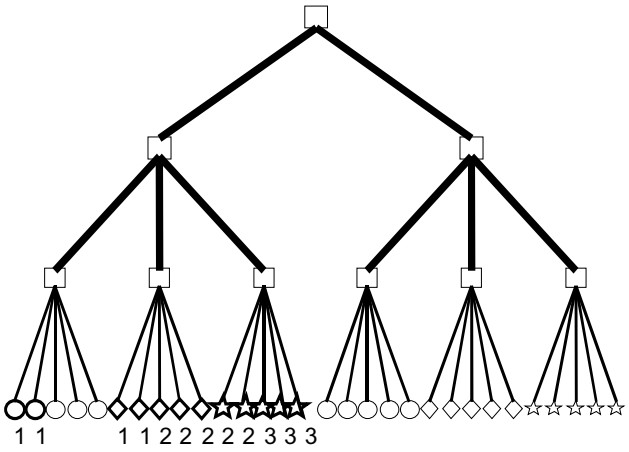

Figure 4. Optimal server assignment in a small service core

6.2. Example 2

The second example is a larger service core that contains 200 server nodes (M = 200), connected via twenty rack switches, four edge switches and one switch mesh. Since it is impossible to show the whole service core in detail, only part of it is displayed in Figure 5, containing the first five rack switches under the first edge switch. Each rack switch is connected to ten server nodes. What is different here from the first example is that the servers under each rack switch may or may not belong to the same server class. For instance, the 3rd rack switch is connected to 5 class-1 servers and 5 class-2 servers, and similarly for the 4th rack switch. The server nodes that are absent under the 1st and 2nd rack switches are servers that are already assigned to another application. So this simulates the deployment of subsequent applications. The particular application has three tiers that require 5, 7, and 5 servers, respectively.

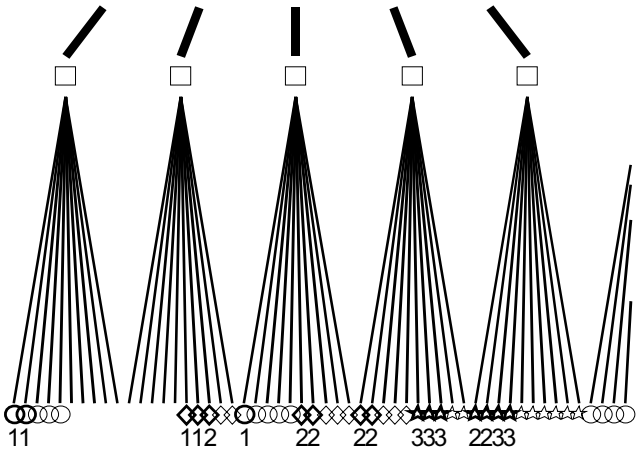

Figure 5. Suboptimal server assignment in a large service core

In this example, the search space becomes too large. Even the LPP algorithm fails to find a global optimum in a reasonable time period. Therefore, local clustering is applied, and the local optimum found inside the first cluster is demonstrated in Figure 5. Again, servers in the 2nd tier are mixed with servers in the 1st or the 3rd tier in most of the racks. The reason for the servers to spread out under various rack switches is the bandwidth constraints on the links that connect rack switches to edge switches.

In general, there is no guarantee that the above local optimum is equal to the global optimum. Unfortunately, our algorithm does not provide an estimate of this difference. An alternative is to search through all the local clusters and use the overall optimum as the solution, which requires longer computation time. However, as we discussed before, it suffices to look at one cluster when there is a lot of symmetry in the topology. It is exactly the case for this particular example, since it is the second application to be deployed. In fact, to validate this claim, we computed the local optima for all the clusters and our original solution turned out to be one of the best.

7. Conclusions and Future Research

The automation and optimization of resource management in large-scale Internet systems have become increasingly important to the successful operation of such systems. In this paper a particular resource assignment problem in a new Internet data center environment consisting of service cores is formulated as a constrained optimization problem. The nonlinearity in both the objective function and the constraints prevents the direct use of conventional linear programming and integer programming packages. However, the special structure of the service core enabled us to develop specific techniques for solving the problem, including projection of the solution set onto a smaller set and a layered partition and pruning scheme to further reduce the amount of the computation. Local clustering is employed to find suboptimal solutions for more complex problems. The effectiveness of these techniques and the impact of parameters are discussed together with some numerical experiment results. In the end, two examples are shown to demonstrate how different algorithms can be used to solve problems of different scale and configuration.

It will be interesting to compare our algorithm with other heuristic search methods, such as Tabu search, simulated annealing, or genetic algorithms. In addition, physical implementation in a laboratory test bed is desirable to study the practical applicability of our approach. Finally, there can be many extensions to the resource and application models in this paper. For example, not all applications have a tiered structure. If we instead consider a general distributed application with a set of servers with certain communication requirements, then the three assumptions we made on the link bandwidth requirements can be removed. Our mathematical model still applies, except that the traffic matrix E will be an arbitrary matrix with a much higher dimension. The specific algorithm in this paper may not be directly applicable in this case. However, similar ideas may be exploited to develop efficient algorithms for the new problem. All these are open questions that need further investigation in our future research.

References

[1] G. Banga, P. Druschel and J.C. Mogul, "Resource containers: A new facility for resource management in server systems," *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation,* New Orleans, Feb. 1999.

[2] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik and P. Wong, "Theory and practice in parallel job scheduling," *Proceedings of IPPS/SPDP '97 Workshop. Lecture Notes in Computer Science*, April 1997, vol. 1291, pp. 1-34.

[3] Global Grid Forum, http://www.gridforum.org/.

[4] V. Kotov and H. Trinks, "Optimization of e-services solutions with the Systems of Servers Library," *Proceedings of MASCOTS 2000*, San Francisco, Aug.-Sep., 2000, pp. 575-582.

[5] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs for multiprocessors," *ACM Computing Surveys*, Dec. 1999, vol. 31(4), pp. 406-471.

[6] D. Menasce, V. Almeida, R. Riedi, R. Flávia, 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.

[7] J. Rolia, S. Singhal and R. Friedrich, "Adaptive Internet Data Centers," *Proceedings of SSGRR 2000 Computer and eBusiness Conference*, L'Aquila, Italy, July-Aug., 2000.

[8] H. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. on Software Engineering*, Jan. 1977, vol. SE-3, no. 1, pp. 85-93.

[9] B.-R. Tsai and K.G. Shin, "Communication-Oriented assignment of task modules in hypercube multicomputers," *Proceedings of the 12th International Conference on Distributed Computing Systems*, June 1992, pp. 38-45.