Server Allocation Problem for Multi-Tiered Applications

Kamalika Chaudhuri, Anshul Kothari, Ram Swaminathan,
Robert Tarjan, Alex Zhang, Yunhong Zhou
HP Laboratories Palo Alto
HPL-2004-151
September 8, 2004*

E-mail: {firstname.lastname}@hp.com

capacity planning, resource allocation, response time, approximation algorithm, knapsack problem

Last few years have seen exponential growth in the area of web applications, especially, e-commerce and web services. One of the most important QoS metric for web applications is the response time for the user. Web application normally has a multi-tier architecture and a request might have to traverse through all the tiers before finishing its processing. Therefore, a request's total response time is the sum of response time at all the tiers. Since the expected response time at any tier depends upon the number of servers allocated to this tier, many different configurations (number of servers allocated at each tier) can give the same QoS guarantee in terms of total response time. Naturally, one would like to find the configuration, which minimizes the total system cost and satisfies the total response time guarantee. Zhang et al. [15] have modeled this problem as a non-linear integer optimization problem and proposed heuristics to solve it optimally.

In this paper we study computational complexity of this non-linear optimization problem, which we call the *multi-tier problem*. We show, for the case of variable number of tiers, the decision version of this problem is NP-Complete and present efficient approximation algorithms (2-approximation in linear time and fully polynomial time approximation scheme) .For the case of constant number of tiers, we show that the problem can be solved in polynomial time.

* Internal Accession Date Only Approved for External Publication

Server Allocation Problem for Multi-Tiered Applications

Kamalika Chaudhuri Anshul Kothari Ram Swaminathan Robert Tarjan
Alex Zhang Yunhong Zhou*

HP Labs, 1501 Page Mill Rd, Palo Alto, CA, 94304

{firstname.lastname}@hp.com

Abstract

Last few years have seen exponential growth in the area of web applications, especially, e-commerce and web services. One of the most important QoS metric for web applications is the response time for the user. Web application normally has a multi-tier architecture and a request might have to traverse through all the tiers before finishing its processing. Therefore, a request's total response time is the sum of response time at all the tiers. Since the expected response time at any tier depends upon the number of servers allocated to this tier, many different configurations (number of servers allocated at each tier) can give the same QoS guarantee in terms of total response time. Naturally, one would like to find the configuration, which minimizes the total system cost and satisfies the total response time guarantee. Zhang et al. [15] have modeled this problem as a non-linear integer optimization problem and proposed heuristics to solve it optimally.

In this paper we study computational complexity of this non-linear optimization problem, which we call the *multi-tier problem*. First we show, for the case of variable number of tiers, the decision version of this problem is *NP-Complete*. Then we present a simple two-approximation algorithm which runs in linear time and a fully polynomial time approximation scheme. For the case of constant number of tiers, we show that the problem is polynomial time solvable.

1 Introduction

Last few years have seen a tremendous growth in the area of web-applications such as electronic commerce, web service, search engines etc. As these applications are user oriented, their main objective is to keep their users satisfied by means of meeting certain quality of service guarantees. One of the most important quality of service parameters for these applications is *expected response time*, which is the total time it takes a user's request to be processed.

Most of the web-applications have a multi-tiered architecture, where a user's request is processed by multiple (levels of) servers. For example, a typical web-service system can be thought to consist of three tiers: web servers, application servers and database servers. Web servers talk to the user and serve as an interface between application servers and the user. Application servers parse users' requests and possibly do some intensive computations. Application servers also needs to talk to back-end database servers to obtain user profiles and system data. Within each tier, multiple machines can be provisioned to share the incoming workload which consists of a series of different types of web requests. At each of the tiers, a user's request is going to suffer *queuing delay* and *processing delay* and the expected response time bounds the total delay suffered by a user. The response time is the time taken for a web request to go through the three-tiered system (i.e. "server-side response time").

Even though three tiers are typical for a web-application architecture, it is possible for the application to have more tiers. Consider a typical search engine such as google [2], the application server tier is actually very complex, and it consists of multiple sub-tiers, working on things such that crawling, parsing, indexing, ranking, searching etc. If the service demand for one tier is high, normally a few load balancing servers are placed in front of the tier to divide the load equally into multiple servers, and they act as one tier also.

*Corresponding author.

Our question is to allocate an adequate number of servers to each tier in order to meet certain service level requirement. Applications that allow different number of machines at each tier are called *horizontally scalable*. We deal with such a horizontally scalable system where service level requirement is expressed in average response time of the system. At each tier, the queuing delay suffered by a request is a function of the number of servers available at that tier. One can reduce the total delay by increasing the number of servers, however, it also increases the total infrastructure cost. Therefore, there is a tussle between the infrastructure cost and total delay and in turn with expected response time. This necessitates the need for a tool, which given the application's workload and expected response time, finds a server allocation (how many servers at each tier) with minimum cost.

Since the system response time is the sum of response times at each of the tiers, the total response time requirement can be met by different configurations in the number of machines at each tier. For example, the configuration of 3 web servers, 2 application servers, and 1 database server may achieve the same average response time as the configuration of 1 web server, 3 application servers, and 1 database server. The question, then, is to find a minimum number of machines in total to meet the average response time guarantee. Realizing that machines at different tiers might be different and incur different costs, the more general optimization problem is to minimize the total weighted sum of servers, with the weights reflecting the "total cost of ownership" (including various costs such as hardware and software costs, operating costs, etc) of servers at different tiers.

In this paper, we study this server allocation problem by modeling it as a non-linear integer optimization problem. For the case where the application has arbitrary number of tiers, we show that the problem is NP-hard. We then present a simple two-approximation algorithm based on Lagrangian relaxation. Next, we present a pseudo polynomial time algorithm and a fully polynomial time approximation scheme (FPTAS). Finally, for the case where the application has constant number of tiers, we show that the problem can be solved in polynomial time. We also observe that Lenstra's algorithm [12] for integer linear programming with a constant number of variables can be extended to an arbitrary convex body with the existence of a separation oracle.

1.1 Related Work

This problem is motivated by TAO (web transaction and optimization), a HP project focusing on developing metrics, models, and infrastructures to effectively manage the performance of web applications. Please refer to Garg et al. [4] for a more detailed discussion of the system and modeling efforts. Appleby et al. [1] describe various aspects of IBM's Océano project which is centered on service level management and which includes many issues that the TAO project also sets out to solve. Menasce and Almeida [14] present general issues in capacity planning issues for web performance.

Zhang et al. [15] have modeled the multi-tier problem as a non-linear integer optimization problem and proposed heuristics to solve it optimally. Zhu et al. [16] addressed the issue of allocating resources (machines) in a tree-like topology of a data center, considering performance constraints such as link bandwidth and switch capacity while minimizing communication delay between the assigned servers. They propose a mathematical optimization model with binary variables for optimally configuring the topology. Our work differs from [16] in several important ways. First, we consider only the average response time performance measure. Second, our topology is a tiered structure compared to an arbitrary topology. The use of the queuing model in expressing the average response time leads to a general non-linear integer problem, but with only a few general integer variables. These simplifications allow us to devise efficient algorithms to find near optimal solutions in polynomial time.

We point out that the solution to the server allocation problem is relevant to the dynamic resource allocation at a Utility Data Center (UDC). The dynamic resource allocation at a UDC requires the integration of demand and capacity planning, and this concept is known as *Capacity on Demand*. Capacity on Demand consists of optimizing the assignment of shared resources to satisfy the requirements of multiple applications. A UDC enables multiple applications to be hosted on a collection of shared resources, where the resources assigned to the applications may be increased when workload increases and reduced when workload reduces. This dynamic resource allocation allows flexible service level agreements (SLAs) in an environment where peak workload is much greater than the normal steady state. A key problem with Capacity on Demand is that the user needs are translated into a logical configuration, and physical resources are then assigned to the logical configuration to satisfy the resource requirements of the application. The problem addressed in

this paper is precisely to determine the optimal resource requirements of an application under a workload in such a way that application SLAs are satisfied.

1.2 Organization of the Paper

The rest of the paper is organized as follows. We describe the queuing model which gives the expected (average) response time as a function of demand and the amount of resources allocated in Section 2.1. In Section 2.2, we will then formulate the multi-tier problem as a mathematical optimization problem consisting of a linear objective function and a nonlinear constraint, with general integer decision variables. In Sections 3 we show that the multi-tier problem is NP-complete. In Section 4, we present the optimal fractional solution using Lagrangian relaxation and give a two-approximation algorithm based on rounding up the optimal fractional solution. In Section 5, we give a pseudo-polynomial time algorithm and as well an $(1+\epsilon)$-approximation algorithm in polynomial time. In Section 6, we show that the problem is polynomial time solvable if the number of tiers is a constant.

2 Problem Formalization

We now describe the response time model, and using this model, then state the multi-tier problem precisely.

2.1 The Response Time Model

One of the nice properties of multi-tier architecture is that the delay suffered by a request in a tier only depends upon the number of servers in that tier and is not affected by how many servers we have at any other tier. Therefore, one can compute request's response time by computing the delays at individual tiers and then summing them up to obtain the total response time.

To simplify our modeling effort for each tier, we assume that all servers at the same tier are identical, and that the workload is shared approximately equally among all the servers at the same tier. If the request arrival rate is λ_i for the i-th tier with N_i servers, then each server has a request arrival rate of λ_i/N_i. Each server is modeled as a processor sharing queue. The expected response time is given by the expected time in system with M/G/1[1] queuing model:

$$R_i(N_i) = \frac{E[S_i]}{1 - (\frac{\lambda_i}{N_i})E[S_i]} \ ,$$

where $E[S_i]$ is the expected processing time (or service demand) of a request on the critical resource (such as CPU) at the i-th tier.[2] For a reference to general queuing theory and the above formula, please see Kleinrock [10]. As discussed before, the response time, $R(N)$, for a k-tier application is the sum of the delays in all the tiers. Therefore

$$R(N) \ = \ \sum_{i=1}^{k} R_i(N_i) \ = \ \sum_{i=1}^{k} \frac{E[S_i]}{1 - \frac{\lambda_i E[S_i]}{N_i}}, \tag{1}$$

where $N = (N_1, N_2, \ldots, N_k)$. We refer to N as a *configuration*.

2.2 The Multi-Tier Problem

From Eq. (1), it follows that there exist multiple server allocations, which have the same response time, and among these feasible allocations, one would like to find the one, which yields the minimum cost. This can be formulated as the following optimization problem:

[1] Poisson distribution for request interarrival density, arbitrary service processing time distribution, one single queue with unbounded buffer length.

[2] $E[S_i]$ can be estimated from the measured utilization rate, u_i, of the critical resource as follows: $E[S_i] = u_i/(\lambda_i/N_i)$.

$$\min_{N_i} \quad \sum_{i=1}^{k} h_i N_i \tag{2}$$

$$\text{s.t.} \quad R(N) = \sum_{i=1}^{k} \frac{E[S_i]}{1 - \frac{\lambda_i E[S_i]}{N_i}} \leq T_0; \tag{3}$$

$$N_i \text{ integer with } N_i > \lambda_i E[S_i], \text{ for } i = 1, \cdots, k,$$

where T_0 is the guaranteed response time and weight h_i's (all assumed to be strictly positive) reflect the difference in the costs of the different servers. [3] Because

$$\frac{E[S_i]}{1 - \frac{\lambda_i E[S_i]}{N_i}} = \frac{N_i E[S_i]}{N_i - \lambda_i E[S_i]} = E[S_i] + \frac{\lambda_i E[S_i]^2}{N_i - \lambda_i E[S_i]}, \quad \text{for all } i,$$

the non-linear constraint (3) can be further simplified as follows:

$$\sum_{i=1}^{k} \frac{\lambda_i E[S_i]^2}{N_i - \lambda_i E[S_i]} \leq T_0 - \sum_{i=1}^{k} E[S_i].$$

Let $a_i = \lambda_i E[S_i]^2$, $b_i = \lambda_i E[S_i]$ and $T = T_0 - \sum_{i=1}^{k} E[S_i]$, then the response time constraint becomes:

$$\sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T. \tag{4}$$

Given this formulation of the optimization problem, one can define the corresponding decision problem as follows: for a given cost p, does there exist an allocation $N = (N_1, \ldots, N_k)$ such that,

$$\sum_{i=1}^{k} h_i N_i \leq p$$

$$\sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T$$

$$N_1, N_2, \cdots, N_k \text{ positive integers with } N_i > b_i, \forall\ i.$$

For simplicity, we assume that h_i's are positive integers for all i. However, a_i, b_i are not necessarily integers. It is easy to see that if one can solve the decision problem in polynomial time then the optimization problem can also be solved in polynomial time by doing a binary search on possible values of p. In the next section, we show that for arbitrary k, solving the decision problem is *NP-Complete*.

3 NP-Completeness

It is tempting to solve the multi-tier problem optimally. However, the following result shows that the problem is "hard" to solve optimally in polynomial time.

Theorem 1 *With arbitrary number of tiers, the multi-tier decision problem is NP-complete.*

Proof: It is easy to see that the multi-tier decision problem is in NP, since a non-deterministic algorithm need only guess a feasible solution and check in linear time whether these two constraints (cost constraint

[3] In the case where costs are difficult to ascertain, a convenient simplification would be to minimize the total number of servers ($\sum_{i=1}^{k} N_i$); that is, to set $h_i = 1$ for all i.

and response time constraint) are satisfied. Any feasible solution has its size polynomially bounded by the input size, since $N_i \leq p/h_i$ for all i.

In order to show that the multi-tier decision problem is *NP-Complete*, we reduce the subset sum problem to it. The subset sum problem is one of the original problems shown to be NP-complete by Karp [8].

Subset Sum: Given an instance (I, C) where $I = \{w_i \mid i = 1, \ldots, k\}$ is a set of positive integers and C is a constant, decide whether there exists a subset $S \subset I$ such that $\sum_{i \in S} w_i = C$.

Without loss of generality, we assume that $w_i \geq 4$, for each $i = 1, \ldots, k$, for otherwise, we can multiply each w_i and C by 4.

An instance of multi-tier decision problem can be expressed in terms of a_i, b_i, h_i, for $i = 1, \ldots, k$, and T, p. Given an instance (I, C) of the subset sum problem, we construct an instance of multi-tier decision problem as follows. For the i-th tier, we compute the input a_i and b_i as follows:

$$\frac{a_i}{1-b_i} - \frac{a_i}{2-b_i} = w_i \quad \text{and} \quad \frac{a_i}{2-b_i} - \frac{a_i}{3-b_i} = 1.$$

The above equations uniquely solve a_i, b_i as positive fractionals:

$$a_i = \frac{2w_i(w_i+1)}{(w_i-1)^2} \quad \text{and} \quad b_i = \frac{w_i - 3}{w_i - 1}.$$

Because $w_i \geq 4$ for each i, $b_i \in (0, 1)$ in this construction. Next, let c_i denote the response time for the i-th tier if there is only one server, or equivalently, $c_i = a_i/(1-b_i) = w_i(w_i+1)/(w_i-1)$. Using c_i we define T to be $T = \sum_{i=1}^{k} c_i - C$. In addition, we set $h_i = w_i$ and $p = \sum_{i=1}^{k} w_i + C$.

We next show that the multi-tier decision instance is equivalent to the subset-sum instance. It is easy to show the equivalence from the subset-sum instance to the multi-tier instance. Given a subset-sum solution S with $\sum_{i \in S} w_i = C$, choose $N_i = 2$ for $i \in S$ and $N_i = 1$ otherwise. Now

$$\begin{aligned}
\sum_{i=1}^{k} \frac{a_i}{N_i - b_i} &= \sum_{i=1}^{k} \frac{a_i}{1-b_i} - \sum_{i \in S} \left(\frac{a_i}{1-b_i} - \frac{a_i}{2-b_i}\right) = \sum_{i=1}^{k} c_i - \sum_{i \in S} w_i = \sum_{i=1}^{k} c_i - C = T, \\
\sum_{i=1}^{k} h_i N_i &= \sum_{i=1}^{k} w_i N_i = \sum_{i=1}^{k} w_i + \sum_{i \in S} w_i = \sum_{i=1}^{k} w_i + C = p.
\end{aligned}$$

To show the other way, consider a feasible configuration N for the multi-tier instance. As N is feasible, we have $\sum_i w_i N_i \leq p$ and $\sum_i a_i/(N_i - b_i) \leq T$. Let S be the set of tiers with at least two servers. The way we have chosen a_i and b_i, one can easily see that the response time for the i-th tier is c_i if there is only one server or at least $c_i - w_i - (N_i - 2)$ if there are at least two servers. Therefore, we have:

$$\sum_{i=1}^{k} c_i - \sum_{i \in S} (w_i + N_i - 2) \leq \sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T = \sum_{i=1}^{k} c_i - C,$$

$$\text{i.e.,} \quad C \leq \sum_{i \in S} w_i + \sum_{i \in S} (N_i - 2).$$

Also as $\sum_{i=1}^{k} w_i N_i \leq p = \sum_i w_i + C$, we have $\sum_i w_i(N_i - 1) \leq C$, i.e.,

$$\sum_{i \in S} w_i + \sum_{i \in S} w_i (N_i - 2) \leq C.$$

The above two inequalities combined together imply that

$$\sum_{i \in S} w_i (N_i - 2) \leq \sum_{i \in S} (N_i - 2), \quad \text{i.e.,} \quad \sum_{i \in S} (w_i - 1)(N_i - 2) \leq 0.$$

Because $N_i \geq 2$ for all $i \in S$ and $w_i \geq 4$ for all i, this is only possible if either S is empty or $N_i = 2$ for each element $i \in S$. The former case is not possible as it contradicts the fact that N is a feasible configuration. Therefore, for all elements in S, $N_i = 2$ and we have, $\sum_{i \in S} w_i = C$. ■

4 A Two Approximation Algorithm

In the previous section, we have shown that the multi-tier problem is *NP-Complete* for arbitrary k. Given the hardness of computing the optimal solution one would like to know if it is possible to compute a good approximate solution efficiently. It turns out that a fully polynomial time approximation scheme (FPTAS) exists for our problem. Before presenting the relatively complex approximation scheme, we first give a simple approximation algorithm which runs in linear time. Our simple approximation algorithm guarantees a worst-case performance factor of two, and it will be used for the construction of our FPTAS in Section 5.

The two-approximation algorithm is based on rounding up the optimal fractional solution, which can be computed using Lagrangian relaxation. By relaxing the constraints that N_i's have to be positive integers, we replace them by $N_i > 0$ for all i. Afterwards, we have only one non-linear constraint left, the total response time requirement. We can use Lagrangian multiplier method to incorporate the response time constraint into the objective function and even get a closed form solution for the continuous-variable optimization problem. Recall that our optimization problem is:

$$\min \quad \sum_{i=1}^{k} h_i N_i \tag{5}$$

$$\text{s.t.} \quad \sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T; \tag{6}$$

$$N_1, N_2, \cdots, N_k \text{ are positive integers.}$$

Consider the Lagrangian function where λ is the Lagrangian multiplier:

$$L(N_1, N_2, \cdots, N_k, \lambda) = \sum_{i=1}^{k} h_i N_i + \lambda \left(\sum_{i=1}^{k} \frac{a_i}{N_i - b_i} - T \right).$$

Taking partial derivative with respect to N_i and setting it to zero, we obtain:

$$\frac{\partial L}{\partial N_i} = h_i - \lambda \cdot \frac{a_i}{(N_i - b_i)^2} = 0, \quad i = 1, \ldots, k.$$

Let N_i^f be the optimal fractional value, thus

$$N_i^f = b_i + \sqrt{\frac{\lambda a_i}{h_i}}, \quad i = 1, \ldots, k.$$

Since N_i^f's are continuous, the non-linear constraint must be binding, that is,

$$\sum_{i=1}^{k} \frac{a_i}{N_i^f - b_i} = T.$$

Substituting N_i^f into the equality and simplify algebraically, we can solve for λ:

$$\sqrt{\lambda} = \frac{\sum_{i=1}^{k} \sqrt{h_i a_i}}{T}.$$

And the cost of the relaxed Lagrangian solution becomes

$$\text{cost}(N^f) \; = \; \sum_{i=1}^{k} h_i N_i^f \; = \; \sum_{i=1}^{k} h_i b_i + \frac{(\sum_{i=1}^{k} \sqrt{h_i a_i})^2}{T}.$$

The optimal fractional solution N^f can be converted into a feasible 2-approximation directly. Let N^r be the integer solution got by rounding up N^f. Specifically, $N_i^r = \lceil N_i^f \rceil$, for all i. Because $N_i^r \geq N_i^f$ for all i,

$\sum_{i=1}^{k} a_i/(N_i^r - b_i) \leq \sum_{i=1}^{k} a_i/(N_i^f - b_i) = T$. Therefore N^r satisfies the total response time constraint and N^r is a feasible solution to the multi-tier problem. Next, we show that the N^r is also a 2-approximation. Because $N_i^r = \lceil N_i^f \rceil$, it is easy to see that $1 \leq N_i^r < N_i^f + 1$, for all i.

$$\text{cost}(N^r) \;=\; \sum_{i=1}^{k} h_i N_i^r \;\leq\; \sum_{i=1}^{k} h_i (N_i^f + 1) \;=\; \text{cost}(N^f) + \sum_{i=1}^{k} h_i. \tag{7}$$

Since N^f is optimal fractional solution, its cost is going to be no more than the optimal integral solution, N^*, i.e., $\text{cost}(N^f) \leq \text{cost}(N^*)$. Also, since any optimal integral solution should at least contain one server in each of the tiers, we have $\sum_i h_i \leq \text{cost}(N^*)$. Eq. (7) together with the above two inequalities implies that $\text{cost}(N^r) \leq 2\text{cost}(N^*)$. In summary, we have the following theorem:

Theorem 2 *For the multi-tier problem with k tiers where k is an arbitrary integer variable, we can compute a two-approximation in time $O(k)$.*

It is easy to construct pathological examples where the performance ratio of the above algorithm versus the optimal is arbitrarily close to two, and so our analysis is tight. However, in practice where T is small, we expect the algorithm to work very well, with its performance ratio close to 1. This is because Eq. (7) together with the fact $\text{cost}(N^f) \leq \text{cost}(N^*)$ leads to

$$\frac{\text{cost}(N^r)}{\text{cost}(N^*)} \;\leq\; \frac{\text{cost}(N^r)}{\text{cost}(N^f)} \;\leq\; 1 + \frac{\sum_{i=1}^{k} h_i}{\text{cost}(N^f)}.$$

When T approaches 0, $\text{cost}(N^f)$ approaches to ∞, thus the last term of the above inequalities approaches 0. In other words, when T is very small, the performance of the above algorithm is almost optimal.

5 Fully Polynomial Time Approximation Scheme

In this section, we present our fully polynomial time approximation scheme. We start by transforming multi-tier optimization problem into an equivalent multiple-choice knapsack problem. By bounding the size of each class of items, first we are able to give a pseudo-polynomial time algorithm for our problem. This solution in turn is adapted to design the FPTAS using standard scaling technique.

The multi-choice knapsack problem is a generalization of the ordinary knapsack problem, where there are m sets of items $S_1, \ldots, S_m$ with $|S_i| = n_i$ for all i and $\sum_{i=1}^{m} n_i = n$. Each item $j \in S_i$ consists of a weight w_{ij} and a profit p_{ij}, and we are given a knapsack with capacity W. The objective is to pick exactly one item from each set, such that the profit sum is maximized and the weight sum is bounded by the knapsack capacity W. (See Martello and Toth [13] and Keller et al. [9] for an extensive treatment of knapsack problems.)

The multi-choice knapsack problem has a lot of applications and the multi-tier problem is one of the latest examples. Because the multi-choice knapsack problem contains the knapsack problem as a special case, and the knapsack problem in turn contains the subset sum problem as a special case, it follows that multi-choice knapsack problem is also NP-complete. An approximation algorithm with performance guarantee 5/4 is given by Gens and Levner [5]. A pseudo-polynomial time algorithm using dynamic programming is easy to design. The first fully polynomial time approximation scheme for the multi-choice knapsack problem has been given by Chandra, Hirschberg and Wong [3]. An improved FPTAS is given in the book [9] where $(1+\epsilon)$-approximation is given using standard scaling with time complexity $O(nm \cdot 1/\epsilon)$.

We transform the multi-tier problem into the multi-choice knapsack problem and use the pseudo-polynomial time algorithm for the multi-choice knapsack problem to solve our problem optimally. Given a multi-tier problem instance with k tiers, the multi-choice knapsack problem consists of k sets of items $S_1, \ldots, S_k$. For an item $j \in S_i$:

$$\text{weight}(j) \;=\; \frac{a_i}{j - b_i}, \quad \text{cost}(j) \;=\; j \cdot h_i.$$

For the multi-tier optimization problem, we don't have any cost constraint, thus any $j > b_i$ is a feasible item in S_i. In order to bound the size of S_i, we bound the number of servers needed at each tier. Consider the response time constraint:

$$\frac{a_i}{N_i - b_i} \leq \sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T$$
$$\Rightarrow \quad N_i \geq b_i + \frac{a_i}{T} \equiv n_i^l.$$

Thus, n_i^l is a lower bound on the number of servers needed at the i-th tier.

Next, let us consider the 2-approximation solution N^r. Let $C^r = \text{cost}(N^r)$. Since C^r is an upper bound on total cost of the optimal, we have:

$$h_i N_i + \sum_{j|j\neq i} h_j n_j^l \leq \sum_{j=1}^{k} h_j N_j \leq C^r$$
$$\Rightarrow \quad N_i \leq \frac{C^r - \sum_{j|j\neq i} h_j n_j^l}{h_i} \equiv n_i^u.$$

Thus, n_i^u is an upper bound on the number of servers needed at i-th tier.

Given n_i^l and n_i^u, we only need to consider item $j \in S_i$ satisfying $\lceil n_i^l \rceil \leq j \leq \lfloor n_i^u \rfloor$. Without loss of generality, we assume that n_i^l and n_i^u are both integers. Otherwise we can always replace n_i^l by $\lceil n_i^l \rceil$ and replace n_i^u by $\lfloor n_i^u \rfloor$. Therefore now set S_i has only $n_i^u - n_i^l + 1$ elements. Next we describe our pseudo-polynomial time algorithm to solve the multi-choice knapsack problem.

The multi-choice knapsack problem can be solved, in pseudo polynomial time, by building a *dynamic programming* (DP) table. There are two ways to build the dynamic programming table; either using the cost or using the weight. As the weights of items in our case are not integers, and the costs are integers, we build it using the cost. Let $F(i, c)$ denote the minimum weight of items selected from the first i item sets with total cost bounded by c. Following is the recursion function to build the DP table $F(,)$:

$$F(i, c) = \min_{j \in S_i} \{F(i-1, c - \text{cost}(j)) + \text{weight}(j)\}$$

From Theorem 2, C^r (the cost of N^r) is within twice of cost(N^*), the optimum cost. We restrict the size of the cost parameter (number of columns) of the table to $B = C^r - \sum_{i=1}^{k} h_i n_i^l$. Thus, the total time taken to build the table is $O(B \cdot \sum_{i=1}^{k} (n_i^u - n_i^l))$. Once the table has been built, the optimal solution can be found by going through the last row and choosing the minimum cost c, such that $F(k, c)$ is bounded by at most T. Thus, we have the following result:

Theorem 3 *The multi-tier problem is pseudo-polynomial time solvable. The optimal solution can be computed in time* $O(B \cdot \sum_{i=1}^{k} (n_i^u - n_i^l))$.

The pseudo polynomial algorithm given above can be converted into a fully polynomial time approximation scheme using cost scaling. The following theorem is our main result in this section:

Theorem 4 *For the multi-tier problem with k tiers, we can compute a $(1+\varepsilon)$-approximation with time $O(k^3 \cdot 1/\varepsilon^2)$ and space $O(k \cdot 1/\varepsilon)$.*

Proof: We first scale the cost of each item as follows:

$$\text{scost}(j) = \lceil \frac{k \cdot \text{cost}(j)}{\varepsilon \cdot C^r} \rceil, \quad \forall\ j \in S_i.$$

It is easy to see that, using the scaled cost, the cost of the optimal solution is bounded by $k \cdot 1/\varepsilon$. Let S_i' denote the i-th item set with scaled down cost. If two items have the same scaled cost, we only keep the one with less weight. Because there are only $k \cdot 1/\varepsilon$ different scost values, S_i' should contains at most k/ε different items. For the i-th set S_i, there are $n_i^u - n_i^l + 1$ items before the scaling, which is exponentially large. The

naïve way of computing S'_i (scaling and discarding afterwards) will result in exponential running time and space. In the following we show that it is possible to compute S'_i with both time and space $O(k \cdot 1/\epsilon)$.

Fix i. For $t = 1, \cdots, k/\epsilon$, consider all the items $j \in S_i$ such that $\text{scost}(j) = t$. We can find the maximum value of j in $O(1)$ time. Once we get the maximum $j = j_t$, its weight becomes $\text{weight}(j_t) = j_t \cdot h_i$. We then put the item with scost $= t$ and weight $= \text{weight}(j_t)$ into set S'_i.

In order to save space for the algorithm, we don't compute all the S'_i in advance. Furthermore, when we compute the DP table, we don't need to store the whole table in memory. Instead, we compute $F(,)$ row by row. For $i = 1, \ldots, k$, we compute $F(i, c)$ for all possible values of c. At the time of computing the i-th row $F(i, .)$, we first compute S'_i, then compute $F(i, .)$ using the values of $F(i-1, .)$ and S'_i. After the i-th row $F(i, .)$ is finished, we discard S'_i and $F(i-1, .)$, and continue to process the $(i+1)$-th row. Thus the total storage requirement for our algorithm is just $O(k \cdot 1/\epsilon)$.

For the total processing time, there are k rows of the DP table, each column has length k/ϵ, and each cell takes time $O(k/\epsilon)$ to compute. Thus the total processing time becomes $O(k^3 \cdot 1/\varepsilon^2)$. In summary, we need time $O(k^3 \cdot 1/\epsilon^2)$ and space $O(k \cdot 1/\epsilon)$ for this algorithm. Each item with scaled cost induces an error bounded by $\epsilon \cdot C^r/k$. There are totally k items in the solution, thus the total error for cost is bounded by $\epsilon \cdot C^r$. Since C^r is a two-approximation of the optimum cost, it follows that the total error is bounded by $\epsilon \cdot C^r \leq 2\epsilon \cdot \text{cost}(N^*)$. By replacing ϵ by $\epsilon/2$, it will only change the above analysis by a constant factor, thus this gives a $(1+\epsilon)$-approximation to the multi-tier problem. ■

6 Constant Number of Tiers

In Section 3, we showed that the multi-tier problem is NP Hard for arbitrary number of tiers. However, real world applications are usually composed of only a small number of stages. For example, a typical ecommerce application has only three tiers: a web server tier, an application server tier, and a database server tier. So it is natural to ask if one could do better if the number of tiers is small. It turns out that it is indeed possible to solve multi-tier problem in polynomial time when the number of tiers is constant. This can be done by a variant of Lenstra's algorithm for solving integer linear programs with constant number of variables. For the sake of completeness, we provide the details in this section.

For the rest of the section, we look at the following generalization of multi-tier problem, which we call Generalized Multitier problem: Given a convex body K in n dimensions described by a set of constraints (linear or nonlinear), we would like to determine if there is an integer point inside the body.

6.1 Definitions and Notation

Before we proceed with a description of Lenstra's algorithm, let us look at a few definitions.

Definition 1 *Given a set B of linearly independent vectors $B = \{b_1, b_2, \cdots, b_n\}$, the lattice spanned by B is the set*

$$L = \{\sum_{i=1}^{n} x_i b_i | x_i \in \mathbb{Z}\}.$$

The vectors $\{b_1, b_2, \cdots, b_n\}$ form a *basis* of the lattice. For example, $\mathbb{Z}^n$ is a lattice spanned by the unit coordinate vectors. Given a lattice L, we would like to determine if there is a point in L inside the body K.

Consider the matrix with columns $b_1, b_2, \cdots, b_n$. The absolute value of the determinant of this matrix depends only on the lattice L and not on the particular basis. We call this determinant $det(L)$. We can interpret $det(L)$ as the volume of the n-dimensional parallelepiped formed by the vectors $b_1, b_2, \cdots, b_n$. The volume interpretation tells us that

$$|det(L)| \leq \prod_{i=1}^{n} |b_i| \tag{8}$$

where the equality holds only when the b_i's are orthogonal.

Definition 2 *A basis is called a* reduced basis *when the basis vectors are almost orthogonal. More formally,*

$$\prod_{i=1}^{n} |b_i| \leq c \cdot |det(L)|$$

where c is a constant depending only on the dimension n.

We will make use of the following simple lemma, which states that given any point in $\mathbb{R}^n$, there is a lattice point close enough to it. The lemma is due to [12]. Due to completeness, we also give a proof for it.

Lemma 1 *Let L be a lattice spanned by $\{b_1, b_2, \ldots, b_n\}$, where the basis vectors are ordered in increasing order of magnitude. Then for any point $x \in \mathbb{R}^n$, there is a point $y \in L$ such that*

$$|x - y| \leq \frac{1}{2}\sqrt{n} \cdot |b_n|$$

Proof: The proof is by induction on n, the number of dimensions. It is easy to see that the base case $n = 1$ holds. Suppose the statement holds for an $n-1$ dimensional lattice.

Let h be the distance of b_n from the hyperplane $H = \sum_{i=1}^{n-1} \mathbb{R}b_i$; then we can always find an integer y_n such that $x - y_n b_n$ is at most a distance $\frac{1}{2}h$ from H. Let $x - y_n b_n = x_1 + e$, where $x_1 \in H$ and e is perpendicular to H. We know that

$$|e| \leq \frac{1}{2}h \leq \frac{1}{2}|b_n|. \tag{9}$$

By the induction hypothesis, there is some point y_1 in the lattice $L' = \sum_{i=1}^{n-1} \mathbb{Z}b_i$ such that $|x_1 - y_1| \leq \frac{1}{2}\sqrt{n-1}|b_{n-1}|$; the distance of the lattice point $y_1 + y_n b_n$ from x is therefore at most $\sqrt{\frac{1}{4}(n-1)|b_{n-1}|^2 + \frac{1}{4}|b_n|^2}$ which is at most $\frac{1}{2}\sqrt{n}b_n$. ■

Next, we would need to define a separation oracle for a convex body.

Definition 3 *A separation oracle for a convex body $K \in \mathbb{R}^n$ is an oracle which, given a point $x \in \mathbb{R}^n$,*

1. *either asserts that $x \in K$,*
2. *or finds a hyperplane H such that K and x lie on opposite sides of H.*

6.2 The Algorithm

Lenstra's algorithm [12] works in n phases. Each phase either finds an integer point inside K, or identifies a direction along which the current body is "flat". By "flat", we mean that the coordinate along that direction of any point inside K lies in a constant sized interval. This reduces the problem to a constant number of subproblems in dimension one lower than the dimension of the current problem.

In each phase, we first find an endomorphism τ which makes K look almost spherical. More formally, after applying the transformation, we can find two concentric spheres S_1 and S_2 of radii r and R respectively such that $S_1 \subseteq \tau(K) \subseteq S_2$ and the ratio of the radii of S_2 and S_1 is a constant which depends only on the dimension n. τ will map $\mathbb{Z}^n$ to a lattice L; we compute a reduced basis $\{b_1, b_2, \cdots, b_n\}$ for L. The basis vectors are ordered such that $|b_{i+1}| \geq |b_i|$ for each i.

Let p be the common center of S_1 and S_2. We find q, the lattice point nearest to p. If q lies inside $\tau(K)$, then we have found an integer point inside K. Otherwise, we consider all hyperplanes H_i which intersect S_2 and correspond to $H + yb_n$ where $y \in \mathbb{Z}$, and recurse on each $n-1$ dimensional subproblem of finding a lattice point inside the body $\tau(K) \cap H_i$.

The number of phases n in the algorithm is constant. As Lemma 2 shows, there are only a constant number of hyperplanes H_i we need to consider, so the number of subproblems at each stage is constant. This makes the running time polynomial, even though it is exponential in terms of n.

Lemma 2 *[12] If $q \notin \tau(K)$, then any lattice point inside $\tau(K)$ can have only a constant number of values for its coordinate corresponding to b_n.*

Proof: Consider the $n-1$ dimensional lattice $L' = \sum_{i=1}^{n-1} \mathbb{Z}b_i$ on the hyperplane $H = \sum_{i=1}^{n-1} \mathbb{R}b_i$. If h is the distance of b_n from H, the hyperplanes H_i are all parallel to H and are at the distance h apart. The volume interpretation of $det(L)$ gives us

$$|det(L)| = h|det(L')| \tag{10}$$

Since the vectors $\{b_1, b_2, \ldots, b_n\}$ form a reduced basis of L,

$$\prod_{i=1}^{n} |b_i| \leq c \cdot |det(L)| \leq c \cdot h \cdot |det(L')| \leq c \cdot h \cdot \prod_{i=1}^{n-1} |b_i| \tag{11}$$

This tells us that $|b_n| \leq c \cdot h$, i.e., $1/h \leq c/|b_n|$.

Now if q lies outside K, q must also lie outside the smaller sphere S_1, i.e., $|p - q| > r$. Lemma 1 tells us that $|q - p| \leq \frac{1}{2}\sqrt{n}|b_n|$. These two inequalities implies the following relationship between r and b_n: $r < \frac{1}{2}\sqrt{n}|b_n|$. Combining this fact with Eq. (11) gives us

$$\frac{2R}{h} = \frac{R}{r} \cdot 2r \cdot \frac{1}{h} \leq \frac{R}{r} \cdot \sqrt{n}|b_n| \cdot \frac{c}{|b_n|} = \frac{R}{r} \cdot c \cdot \sqrt{n}. \tag{12}$$

Since the hyperplanes H_i are a distance h apart, there can be at most $2R/h$ such hyperplanes intersecting S_2. Because c is a constant only depending on n, n is a constant, and R/r is bounded by a constant only related to n, Eq. (12) implies that $2R/h$ is a constant only related to n. The proof is complete now. ■

The following two theorems complete the proof that Lenstra's algorithm works correctly.

Theorem 5 *[6] Suppose we have a convex body K in n dimensions specified by a separation oracle. We can find in polynomial time two ellipsoids E_1 and E_2 with the following properties:*

1. $E_1 \subseteq K \subseteq E_2$
2. $E_2 = O(n^{3/2}) \cdot E_1$

We refer to [7] for a detailed description of the algorithm for constructing the ellipsoids. Once we obtain E_1 and E_2, we can take τ to be the transformation which maps E_1 and E_2 to spheres. Designing a separation oracle for our convex body K is easy; for a point that violates a linear constraint, we take the same constraint as the separating hyperplane. For a point p that violates a non-linear constraint, we draw a line between p and some arbitrary point q inside K. Let r be the point where this line intersects K; the tangent at r to the non-linear constraint curve will be our separating hyperplane.

Theorem 6 *[11] Given an arbitrary basis $B' = \{b'_1, b'_2, \ldots, b'_n\}$ of a lattice L, it is possible to find a reduced basis $B = \{b_1, b_2, \ldots, b_n\}$ of L with $c = 2^{n(n-1)/4}$. Moreover, B can be found in polynomial time.*

Combining Theorems 5 and 6 and Lemma 2 leads to the following result.

Theorem 7 *Suppose we have an n-dimensional convex body K described by a separation oracle. If n is a constant, it is possible to determine in polynomial time if there is an integer point inside K.*

Proof: Theorems 5 and 6 and Lemma 2 tell us that if there is a separation oracle for K, then we can determine in polynomial time whether there is an integer point inside K. A separation oracle for a convex body described by a set of linear or non-linear constraints can be the following: It is easy to see if a point is inside K; for a point that violates a linear constraint, we take the same constraint as the separating hyperplane. For a point p that violates a non-linear constraint, we draw a line between p and some arbitrary point q inside K. Let r be the point where this line intersects K; the tangent at r to the non-linear constraint curve will be the separating hyperplane. ■

6.3 Application to Multi-Tier Problem

In this subsection, we show how the techniques discussed so far apply to multi-tier problem. The corresponding body K in the multi-tier problem is formed by the surface $\sum_{i=1}^{n} a_i/(N_i - b_i) \leq T$ and the hyperplane $\sum_i h_i N_i \leq C$. If there is an integer point inside K, there exists a solution for the multi-tier problem with cost $\leq C$.

It is easy to see that the body K in our problem is convex. One can also design a separation oracle for K using the techniques described in the proof of Theorem 7. The following is therefore a corollary of Theorem 7.

Corollary 1 *The multi-tier problem with a constant number of tiers can be solved in polynomial time.*

7 Open Problems

For the special case where $k = 2$, our multi-tier decision problem can be simplified to the following form: Does there exists two positive integers x, y such that $xy \geq T$, and $ax + by \leq c$? According to Corollary 1, this simple yet intriguing decision problem is polynomial time solvable. However, the current solution uses a lot of heavy machinery and techniques from high dimensional geometry. One would eventually like to have a simple solution that uses little of this machinery and specific properties of the surface. Is there such an efficient algorithm? We end with this open question.

References

[1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar. Océano – SLA-based management of a computing utility. In *Proc. 7th IFIP/IEEE Intl. Symp. on Integrated Network Management*, May 2001.

[2] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.

[3] A. Chandra, D. Hirschberg, and C. Wong. Approximate algorithms for some generalized knapsack problems. *Theoretical Computer Science*, 3:293–304, 1976.

[4] P. K. Garg, M. Hao, C. Santos, H.-K. Tang, and A. Zhang. Web transaction analysis and optimization (TAO). In *Proceedings of the 3rd Workshop on Software and Performance*, pages 286–293, 2002.

[5] G. Gens and E. Levner. Approximation algorithms for certain universal problems in scheduling theory. *Soviet Journal of Computers and System Sciences*, 6:31–36, 1978.

[6] J. L. Goffin. Variable metric relaxation methods, part ii: The ellipsoid method. *Mathematical Programming*, 30, 1984.

[7] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[8] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[10] L. Kleinrock. *Queueing Systems*. Vol. II: Computer Applications. John Wiley and Sons, 1976.

[11] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[12] H. W. Lenstra. Integer linear programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[13] S. Martello and P. Toth. *Knapsack Problems - Algorithms and Computer Implementations*. John Wiley and Sons, New York, 1991.

[14] D. A. Menasce and V. A. Almeida. *Capacity Planning for Web Performance*.

[15] A. Zhang, P. Santos, D. Beyer, and H.-K. Tang. Optimal server resource allocation using an open queueing network model of response time. Technical Report HPL-2002-301, HP Labs, 2002.

[16] X. Zhu, C. Santos, J. Ward, D. Beyer, and S. Singhal. Resource assignment for large-scale computing utilities using mathematical programming. Technical Report HPL-2003-243R1, HP Labs, 2003.