Bandwidth-Constrained Allocation in Grid Computing

Anshul Kothari[*], Subhash Suri[**], and Yunhong Zhou [***]

No Institute Given

Abstract. Grid computing systems pool together the resources of many workstations to create a virtual computing reservoir. Users can "draw" resources using a pay-as-you-go model, commonly used for utilities (electricity and water). We model such a system as a capacitated graph, and study a basic allocation problem: given a set of jobs, each demanding computing and bandwidth resources and yielding a profit, determine which feasible subset of jobs yields the maximum total profit.

1 Introduction

Nearly all leading computer hardware vendors (IBM, Sun, Hewlett-Packard) have recently announced major initiatives in on-demand or grid computing. These initiatives aim to deliver computing resources as utilities (electricity or water)—users "draw" computing power or disk storage from a "reservoir" and pay only for the amount they use. Despite their different names (IBM's On-Demand computing, Sun's N1 computing and HP's Adaptive Infrastructure), the motivation behind these technologies is the same: many users (scientific labs, industries) often need extremely high computing power, but only for short periods of time. Examples include software testing of new systems or applications, verification of new chip designs, scientific simulations (geological, environmental, seismic), molecular modeling etc. Building and managing dedicated infrastructure is expensive, especially if its use is sparse and bursty. In addition, a vast amount of computing and disk capacity at enterprises is idle for large fraction of the time. These new initiatives aim to harness this power by creating a virtual computing reservoir.

In an unrelated effort, peer to peer computing (P2P) also envisions a *world wide computer*, which aims to combine all the informational resources of the web (computing power, storage, data) through a loosely coupled network of

[*] Department of Computer Science, University of California, Santa Barbara, CA 93106. Email: kothari@cs.ucsb.edu. Supported in part by National Science Foundation grants IIS-0121562 and CCR-9901958.

[**] Department of Computer Science, University of California, Santa Barbara, CA 93106. Email: suri@cs.ucsb.edu. Supported in part by National Science Foundation grants IIS-0121562 and CCR-9901958.

[***] Hewlett-Packard Laboratories, 1501 Page Mill Road, 5U-64, Palo Alto, CA 94304. Email: yunhong.zhou@hp.com.

peers [7]. The efforts of on-demand or grid computing are aimed at much smaller, enterprise level organizations, where a cluster of workstations and storage devices act as a shared resource pool. These two technologies may appear superficially similar, at least in their current form, yet they differ in some important ways: grid has a centralized administration, P2P does not; grid assumes reliable, robust, trustworthy machines, P2P does not. For the purpose of this paper, we will focus on the "grid computing" model and study a natural algorithmic problem of task scheduling; however, our problem setting applies more broadly.

The current grid systems only provide the CPU or disk units; *there is no bandwidth guarantee.* Many scientific simulations, as well as real-time applications like financial services, involve sustained high data transfer rates, and thus require a guaranteed application level bandwidth. The bandwidth is a different type of resource: it's a *link* resource, whereas computing cycles and disk units are *node* resources. We consider the following natural problem in this setting: given a set of tasks, each requesting some computing and some bandwidth resources and yielding a profit if chosen, determine which subset of jobs yields the maximum profit, given the current resources of the grid. We will only consider the *offline* version of the problem, leaving the online case as a future direction.

We model the resource pool (grid) as an undirected graph $G = (V, E)$, with n nodes and m edges, where each node $v \in V$ has a computing resource $C(v)$, and each link (u, v) has a bandwidth $B(u, v)$. (We assume that the computing resources are expressed in a common unit, such as normalized CPU cycles.) We are given a set of k jobs, $J_1, J_2, \ldots, J_k$. The job J_i is specified by a triple $\langle c_i, b_i, p_i \rangle$, where c_i, b_i are the computing and the bandwidth resource needed by J_i, and p_i is the profit for this job if chosen. Let $C_i(v_k)$ denote the computing resource that v_k contributes to J_i, and let $B_i(u, v) \in \{0, b_i\}$ denote the bandwidth that (u, v) reserves for J_i. If job J_i is accepted, then we must have (i) $\sum_k C_i(v_k) \geq c_i$, namely, c_i units of the computing resource are allocated to J_i, and (ii) the set of edges $\{(u, v) \mid B_i(u, v) = b_i\}$ spans V_i. That is, the set of nodes that contribute computing resources for J_i must be *connected* by a subset of links with *reserved* bandwidth b_i. (Acceptance of a job is a binary decision: either it is accepted, or it is rejected; it cannot be partially accepted.) An index set of jobs $\mathcal{J}$ is *feasible* if neither the computing nor the bandwidth resource capacity is violated, that is, $\sum_{i \in \mathcal{J}} C_i(v_k) \leq C(v_k)$, for all nodes $v_k \in V$, and $\sum_{i \in \mathcal{J}} B_i(u, v) \leq B(u, v)$, for all links $(u, v) \in E$. See Figure 1 for an example. The total profit for the accepted jobs is $\sum_{i \in \mathcal{J}} p_i$. The goal of the allocation problem is to determine the feasible subset of jobs that yields the maximum profit.

Our Results

Without the bandwidth constraint, the allocation problem in the grid computing is the integer knapsack problem: the CPU pool is the knapsack, and each job is an item. Integer knapsack is (weakly) NP-complete, but one can solve it optimally in pseudo-polynomial time. (One can reasonably assume that the total number of computing units is polynomially bounded in n.)

We begin our investigation by studying *when* does the network bandwidth even become a bottleneck in grid computing. To this end, let $b_{\max}$ denote the *maximum bandwidth requested* by any job, and let $B_{\min}$ denote the *minimum capacity* of any link in G. Our first result shows that as long as no job requests more than half the minimum link bandwidth, namely, $b_{\max} \leq \frac{1}{2}B_{\min}$, the bandwidth guarantee can be provided essentially for free (Theorem 1). In this case, therefore, an optimal allocation can be computed in (pseudo) polynomial time.

We next show that $\frac{1}{2}B_{\min}$ forms a sharp boundary: if job bandwidths are even slightly larger than $\frac{1}{2}B_{\min}$, then the allocation problem becomes *strongly* NP-complete. Under the reasonable assumption that $b_{\max} \leq B_{\min}$ (i.e. no link is a bottleneck for any *single* job), we present an efficient approximation scheme that guarantees at least one-third of the maximum profit.

The allocation problem turns out to be hard if we allow $b_{\max} > B_{\min}$; that is, the jobs demand bandwidths in excess of some of the link capacities. In this case, we show that even a *path topology* network is intractably hard. We present an $O(\log B)$ approximation scheme for the path topology, where all the bandwidths requested by the jobs lie in the range $[1, B]$. As part of our path topology solution, we also develop a new algorithm for the strongly NP-complete *multiple knapsack* problem, improving the $(2+\varepsilon)$-approximation scheme of Chekuri and Khanna [3] with running time $O(nk \log \frac{1}{\varepsilon} + \frac{n}{\varepsilon^4})$. Instead, we give a simple 2-approximation algorithm with worst-case running time $O((n+k)\log(n+k))$.

2 Allocation in Grid Computing

The underlying resource pool (grid) is modeled as an undirected graph $G = (V, E)$, with n nodes and m edges, where each node $v \in V$ has a computing resource $C(v)$, and each link (u, v) has a bandwidth $B(u, v)$. A job J_i, for $i = 1, 2, \ldots, k$, is specified by a triple $\langle c_i, b_i, p_i \rangle$, where c_i, b_i are the computing and the bandwidth resource needed by J_i, and p_i is the profit. Let $C_i(v_k)$ denote the computing resource that v_k contributes to J_i, and let $B_i(u, v) \in \{0, b_i\}$ denote the bandwidth that (u, v) reserves for J_i. (Note that computing resources are aggregated across multiple nodes, but the bandwidth resource is binary. Unless a link contributes full b_i units of the bandwidth, it cannot be used for communication between nodes allocated to J_i.) See Figure 1 for an example.

If job J_i is accepted, then we must have (*i*) $\sum_k C_i(v_k) \geq c_i$, namely, c_i total units of the computing resource are allocated to J_i, and (*ii*) the set of edges $\{(u, v) \mid B_i(u, v) = b_i\}$ spans V_i. That is, the set of nodes that contribute computing resources for J_i must be *connected* by a subset of links with *reserved* bandwidth b_i. An index set of jobs $\mathcal{J}$ is *feasible* if neither the computing nor the bandwidth resource capacity is violated, that is, $\sum_{i \in \mathcal{J}} C_i(v_k) \leq C(v_k)$, for all nodes $v_k \in V$, and $\sum_{i \in \mathcal{J}} B_i(u, v) \leq B(u, v)$, for all links $(u, v) \in E$. The total profit for the accepted jobs is $\sum_{i \in \mathcal{J}} p_i$. The goal of the allocation problem is to determine the feasible subset of jobs that yields the maximum profit.

We begin our investigation by asking *when* does the network bandwidth even become a bottleneck. Surprisingly, there turns out to be a rather sharp boundary.

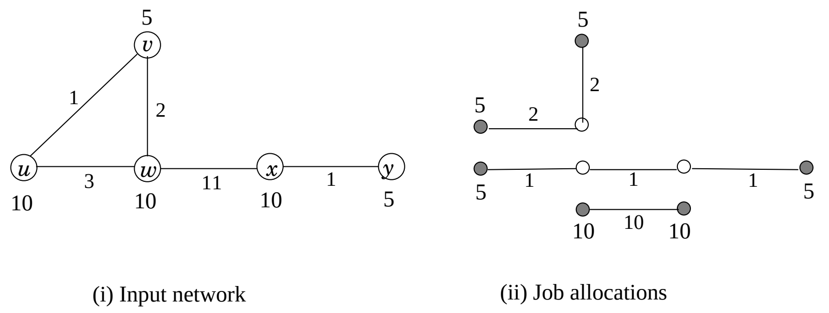

Fig. 1. An example with 3 jobs, $J_1 = \langle 20, 10, p_1 \rangle$, $J_2 = \langle 10, 1, p_2 \rangle$, $J_3 = \langle 10, 2, p_3 \rangle$. Figure (i) shows the input network. Numbers below the nodes denote the resource units available at that node; numbers next to links denote bandwidth. Figure (ii) shows an allocation where all 3 jobs are satisfied; the filled nodes contribute resource units.

Let $b_{\max}$ be the maximum requested bandwidth of any job, and let $B_{\min}$ be the minimum bandwidth of any link in G.

Theorem 1. *Suppose that* $b_{\max} \le \frac{1}{2} B_{\min}$ *holds. Then, the allocation problem can be solved optimally in time* $O(k|C| + n + m)$*, where* $|C|$ *is the total number of computing units available, and* n, m *are the number of nodes and edges in the network. One can also achieve* $(1 + \varepsilon)$ *approximation of the optimal in time polynomial in* $k, 1/\varepsilon$ *and linear in* n *and* m*.*

Proof. We take all the jobs and solve a 0/1 knapsack problem, where we simply aggregate the computing resources of all the nodes in the graph. Job i has size c_i and value p_i; the knapsack capacity is $|C|$. Let W be the set of winning jobs (solution of the knapsack), and let $p(W)$ be their total profit. Clearly, the optimal solution of the resource allocation problem cannot have profit larger than $p(W)$. In the following, we show how to allocate all the jobs of W in G.

Construct any spanning tree T of G. Each link of this tree has capacity at least $B_{\min}$. We root this tree arbitrarily at a node r, and perform a pre-order walk of T. We allocate jobs of W to the nodes encountered in the pre-order; when a node's capacity is depleted, we move to the next node. It is easy to see that no link of the tree is shared by more than 2 jobs, and all the jobs are allocated.

The running time is dominated by the knapsack problem, which takes $O(k|C|)$ time using the dynamic programming. If $(1 + \varepsilon)$ approximation is needed, we can use a fully polynomial approximation scheme, whose running time is polynomial in k and $1/\varepsilon$; the $O(n + m)$ time is for constructing a spanning tree and traversing it. This completes the proof.

Surprisingly, letting the job bandwidth exceed $\frac{1}{2} B_{\min}$ even slightly makes the problem strongly intractable.

Theorem 2. *The optimal allocation problem is* strongly *NP-Complete even if the job bandwidths satisfy the condition* $\frac{1}{2} B_{\min} + \varepsilon \le b_{\max} \le B_{\min}$.

Proof. We reduce the well-known 3-partition problem [8], which is strongly NP-Complete, to our allocation problem. The 3-partition problem is the following:

Instance: Integers m, d and x_i, for $i = 1, 2, \cdots, 3m$ satisfying $\sum_i x_i = md$ and $\frac{d}{4} < x_i < \frac{d}{2} \;\; \forall i$.
Question: Is there a partition of x's into m disjoint (3-element) subsets A_1, A_2, $\cdots$, A_m such that $\sum_{i \in A_j} x_i = d$, for $j = 1, 2, \cdots, m$.

Given an an instance of the 3-partition problem, we construct a tree (of height one) with $3m+1$ nodes $u_0, v_1, \cdots, v_{3m}$. The node u_0 is root and the other $3m$ nodes are its children. The node v_i has x_i units of the resource; the root node has zero units of the resource. Each link has a bandwidth B. We create m identical jobs $\langle d, \frac{1}{2}B + \varepsilon, \; p\rangle$. One can show that all m jobs can be allocated exactly when the input 3-partition instance has a feasible solution.

In the next section, we present a constant factor approximation scheme when $b_{\max} \le B_{\min}$. That is, no network link is a bottleneck for any *single* job. In the subsequent section, we address the general grid model without any constraint on the network link bandwidth.

3 An Approximation Scheme when $b_{\max} \le B_{\min}$

We construct a spanning tree, T, of the input network G, rooted at an arbitrary node r. Since each link of G has bandwidth at least $B_{\min}$, all edges of T have bandwidth at least $B_{\min} \ge b_{\max}$. For a node v, we let T_v denote the subtree rooted at v. Let $C(T_v)$ denote the total (remaining) resource units at the nodes in T_v. That is, $C(T_v) = \sum_{u \in T_v} C(u)$. Similarly, for a subset of nodes $S \subseteq V$, let $C(S)$ denote the total resource units available at the nodes of S. The input set of jobs is $J_1, J_2, \ldots, J_k$. We assume that $c_i \le \sum_{v \in V} C(v)$; otherwise job J_i clearly cannot be satisfied. Our algorithm can be described as follows.

Algorithm Approx

1. Sort the input jobs in descending order of p_i/c_i (profit per compute cycle). Process jobs in the sorted order. Let $J_a = \langle c_a, b_a, p_a\rangle$ be the next job.
2. If $c_a \le C(T_r)$, do Step 3; else do Step 4. (Recall that r is the root of the spanning tree T.)
3. Find the *best fit* node v in the current tree; that is, among all nodes u for which $C(T_u) - c_a \ge 0$, v minimizes $C(T_u) - c_a$.
   – Among the children nodes of v, choose a set S such that $c_a \le C(S) \le 2c_a$. Allocate the set S (and their descendants) to job J_a, and delete these nodes from the tree.
   – If no such S exists, we allocate all the children of v plus the appropriate fraction of v's resources to the job J_a; in this case, we delete all the children of v from T, and update the remaining resource units $C(v)$ for the node v.

   – Add J_a to the set Z, which contains all the accepted jobs.

4. Let $p(Z)$ be the total profit of all the jobs accepted in Z. If $p(Z) \geq p_a$, we output Z, otherwise, we output the single job $\{J_a\}$.

end Algorithm

Theorem 3. *The algorithm* Approx *computes a feasible set of jobs whose profit is at least* $1/3$ *of the optimal. The algorithm can be implemented in worst-case time* $O(m + k \log k + n(k + \log n))$.

Proof. Suppose J_a is the first job that is rejected by the algorithm. Let Z be the current set of accepted jobs when J_a is encountered. Let C_Z be the total number of resource units demanded by jobs in Z; that is, $C_Z = \sum_{i \in Z} c_i$. By the best fit rule, whenever we accept a job in Z, it wastes at most an equal amount of resource. Since J_a could not be allocated, we have the following inequality:

$$2C_Z + c_a > C(T), \tag{1}$$

where $C(T)$ is the total number of resource units initially available in the system. Let d_Z denote the *average unit price* for the jobs in Z. That is,

$$d_Z = \frac{\sum_{i \in Z} p_i}{\sum_{i \in Z} c_i}$$

Let d be the average unit price of $Z \cup J_a$, and let d^* be the average unit price of the jobs in an optimal solution. Since our algorithm considers jobs in the decreasing unit price order, we have $d_Z \geq d \geq d*$. Thus,

$$2p(Z) + p_a = d_Z C_Z + d(C_Z + c_a) \geq d^* C(T) \geq OPT$$

Since our algorithm chooses $\max\{Z, J_a\}$, it follows that $3 \max\{p(Z), p_a\} \geq OPT$. The bound on the worst-case running time follows easily from the description of the algorithm.

The analysis of Approx is tight. The following is an example where the algorithm's output approaches one third of the optimal. Consider the tree network shown in Figure 2. Assume there are 4 jobs. Jobs J_1 and J_2 are $\langle M+\varepsilon, 1, M+2\varepsilon \rangle$, while jobs J_3 and J_4 are $\langle 2M-3, 1, 2M-3 \rangle$. The bandwidth of each link in the tree is also 1. All four jobs can be feasibly allocated, by assigning nodes u, x to J_1, nodes v, y to J_2, node w and half of r to J_3, and node z and half of r to J_4. The total profit is $6M - 6 + 4\varepsilon$.

We now consider the performance of Approx. The algorithm will process jobs in the order $\{J_1, J_2\}, \{J_3, J_4\}$. The algorithm will allocate J_1 to nodes w and x and J_2 to nodes y and z, and will fail to schedule the other jobs. The total profit is $2M + 4\varepsilon$, which approaches $1/3$ of the optimal as M grows.

A natural question is whether the resource allocation problem becomes easier for tree topologies, within the cluster computing model. Unfortunately, that is

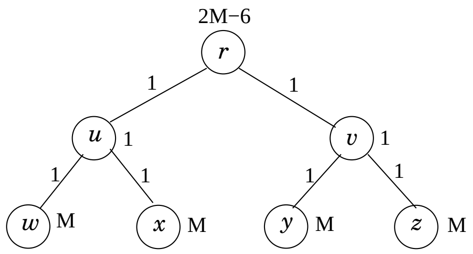

Fig. 2. Tightness of Approx. Nodes u, v have 1 unit of resource; nodes w, x, y, z have M units, and the root has $2m - 6$ units. All links have capacity 1.

not the case, as the reduction of Theorem 2 already establishes the hardness for the trees. If the topology is further restricted to a *path*, however, the problem can be solved optimally in (pseudo) polynomial time.

Theorem 4. *If the network topology is a path and the input satisfies $b_{\max} \leq B_{\min}$, then the allocation problem can be solved optimally in (pseudo) polynomial time.*

4 The Global Grid Model

In the previous section, we assumed that the minimum network link bandwidth is at least as large as the maximum job bandwidth; that is, $b_{\max} \leq B_{\min}$. This is a realistic model for the grid computing at an enterprise level, where a collection of workstations are joined by high bandwidth links. However, when one envisions a larger, Internet scale grid, then this assumption no longer seems justified. In this section, we consider the allocation for this "global grid" model.

Suppose that the link bandwidths are in some arbitrary range $[B_{\min}, B_{\max}]$, and the jobs can request an arbitrary bandwidth (even $b > B_{\max}$); if a job requests bandwidth greater than $B_{\max}$, then it must be allocated to a single node. We call this the *global grid* model for ease of reference. The allocation problem in the global grid appears to be significantly harder than in the previous model. The latter is clearly a special case of the former, and so the intractability theorems of the preceding sections all apply to the global grid as well. In the global grid, however, *even the path topology* is intractable. We use a reduction from the multiple knapsack problem [3], which unlike the single knapsack problem is strongly NP-Complete.

Lemma 1. *The optimal allocation problem in the global grid model is* strongly *NP-complete even if the network topology is a* path.

Thus, the special case of the problem when the network consists of isolated nodes is equivalent to the multiple knapsack problem. We start our discussion with an approximation algorithm for this case.

4.1 Isolated Nodes: 2-Approximation of Multiple Knapsack

Suppose all jobs request bandwidth greater than the maximum link capacity in the network (or, equivalently, if all links have zero bandwidth), then the network reduces to a set of isolated nodes. Our problem is equivalent to the well-known Multiple Knapsack problem. Chekuri and Khanna [3] have given an $O(n^{O(\log(1/\varepsilon)/\varepsilon^8)})$ time approximation scheme for the multiple knapsack problem. They also gave a $(2+\varepsilon)$-approximation scheme with running time $O(nk \log \frac{1}{\varepsilon} + \frac{n}{\varepsilon^4})$. In the following, we show that a simple greedy algorithm achieves a factor 2 approximation in time $O((n+k)\log(n+k))$.

Let $S = \{a_1, a_2, \ldots, a_k\}$ be the set of items, where item a_i has size $s(a_i)$ and profit $p(a_i)$. Given a subset $A \subseteq S$, let $s(A)$ and $p(A)$ denote the total size and total profit of the set of items in A. Let $K = \{1, 2, \ldots, n\}$ be the set of knapsacks, where the jth knapsack has capacity c_j. We assume that that knapsacks are given in *non-decreasing* order of capacity; that is, $c_1 \leq c_2 \leq \cdots \leq c_n$. The items are given in *non-increasing* order of unit price; that is, $p(a_i)/s(a_i) \geq p(a_{i+1})/s(a_{i+1})$.

Algorithm MKP-Approx

```
1. Let $L$ be the list of the remaining items, initialized to $S$.
2. Initialize greedy solution $G = \emptyset$.
3. Consider the knapsacks in sorted order. Let knapsack $j$ be the next one.
   (a) Let $L_j \subseteq L$ be the subset of items such that $s(x) \leq c_j$, for $x \in L_j$.
   (b) Greedily (descending unit price) add items of $L_j$ to the knapsack $j$. Let
       $f_j$ be the first item to exceed the remaining capacity of knapsack $j$.
   (c) Let $A_j \subseteq L_j$ be the set of items that have been added to the knapsack
       when $f_j$ is encountered.
   (d) If $p(A_j) \geq p(f_j)$, add $A_j$ to greedy solution $G$; otherwise add $f_j$ to $G$.
   (e) Remove $A_j$ and $f_j$ from $L$.
4. Return $G$.
```

Due to limited space, we omit the proof of the following theorem.

Theorem 5. *The algorithm* MKP-Approx *achieves a 2-approximation of the Multiple Knapsack Problem in time $O((n+k)\log(n+k))$, where n and k are the number of knapsacks and items.*

4.2 An Approximation Scheme for Path Topology

Our main result for the global grid is an $O(\log B)$ factor approximation scheme, where all jobs have bandwidths in the range $[1, B]$. We begin with some simple observations.

Let $v_1, v_2, \ldots, v_n$ denote the nodes of the path, in the left to right order. Suppose in some allocation v_i (resp. v_j) is the leftmost (resp. rightmost) node contributing the computing resources to a job J. Then, we call $[i, j]$ the *span* of J. We say that two spans $[i, j]$ and $[i', j']$ are *partially overlapping* if they overlap but neither contains the other. In other words, $[i, j]$ and $[i', j']$ partially overlap if $i < i' \leq j < j'$ or $i' < i \leq j' < j$. We say that job $J_1 = \langle c_1, b_1, p_1 \rangle$ is *nested* inside job $J_2 = \langle c_2, b_2, p_2 \rangle$ if the span of J_1 is contained inside the span of J_2. The following two elementary lemmas will be useful in our approximation.

Lemma 2. *There always exists a maximum profit allocation in which no two jobs have partially overlapping spans.*

Lemma 3. *If job $J_1 = \langle c_1, b_1, p_1 \rangle$ is* nested *inside job $J_2 = \langle c_2, b_2, p_2 \rangle$, then $b_1 > b_2$, and there is some link in the span of J_2 whose bandwidth is strictly smaller than b_1.*

We can draw two simple conclusions from the preceding lemmas: (1) if all the jobs require the same bandwidth, then there is an optimal non-nesting solution; and (2) if the maximal bandwidth required by any job is no more than the minimum bandwidth of any link, then again there is an optimal non-nesting solution. In both these cases, we can obtain an optimal (pseudo) polynomial algorithm, using the single 0/1 knapsack solution. In the more general setting, we have the following:

Lemma 4. *If each link in the path network has bandwidth capacity either* 0 *or* B*, then we can get a $(2 + \varepsilon)$-approximation in polynomial time.*

Proof. We partition the input jobs into two classes: *big* jobs, which need bandwidth more than B, and *small* jobs, which need bandwidth at most B. Clearly, the big jobs cannot be served by multiple nodes, while the small jobs can be served by multiple nodes if they are connected with bandwidth B links. Our approximation algorithm works in the following way.

First we consider big jobs and solve it by using the multiple knapsack problem (MKP) with approximation ratio $(1 + \varepsilon/2)$ [3]. We then consider small jobs. The network links with bandwidth 0 partition the path into multiple subpaths, where each subpath is joined by links of capacity B. A small job can only be satisfied by nodes within one subpath. We now consider each subpath as a *bin* with its capacity equal to the sum of capacities for all the nodes contained in it. We apply another $(1 + \varepsilon/2)$-approximation MKP algorithm to this problem and get another candidate solution. Of the two solutions, we pick the one with the larger profit. The following argument shows that this algorithm achieves approximation ratio $(2 + \varepsilon)$.

Consider an optimal solution; it consists of some small jobs and some big jobs. Let Π_s and Π_b, respectively, denote the total profit of the optimal solution contributed by small and big jobs. Thus $OPT = \Pi_s + \Pi_b \leq 2 \max\{\Pi_s, \Pi_b\}$. If A denotes the total profit for our algorithm, then $\Pi_s \leq (1 + \varepsilon/2)A$. Similarly, by considering the large jobs, we get $\Pi_b \leq (1 + \varepsilon/2)A$. By combining these inequalities together, we get $OPT \leq (2 + \varepsilon)A$. This completes the proof.

In order to prove our main result for the path topology in the grid model, we first partition the set of jobs into $\log B$ classes such that each job has roughly the same amount of bandwidth requirement. Let us suppose that all the jobs in the set have their bandwidth requirement between b and $2b$.

Lemma 5. *Suppose that all the jobs have bandwidth requirement in the range* $[b, 2b]$. *The maximum profit realizable by the best nesting solution is at most twice the maximum profit realizable by a non-nesting solution. Thus, limiting our search to the non-nesting solutions costs at most a factor of two in the approximation.*

Proof. Consider an optimal solution for the problem, where jobs may nest arbitrarily with each other. Consider the *containment partial order* among these jobs: $J < J'$ if the span of J is contained in the span of J'; in case of ties, the lower indexed job comes earlier in the partial order. Let s_0 be the set of *maximal* elements in this partial order—these are the jobs whose spans are not contained in any other job's span. Let s_1 denote the set of remaining jobs. Let Π_0 denote the total profit of s_0 in the optimal solution, and let Π_1 be the profit of the s_1 jobs. We argue below that either all jobs in s_0 or all jobs in s_1 can be allocated with non-nesting spans.

The spans of all the jobs in s_0 are clearly non-nesting (by definition). Next, observe that any link that lies in the span of a job in s_1 must have bandwidth at least $2b$, since this link is shared by at least two jobs, and every job has bandwidth at least b. Since the bandwidth needed by any job is at most $2b$, using arguments like the one in Lemma 2, we can re-allocate resources among the jobs of s_1 so that no two jobs nest. Thus, there exist an alternative non-nesting solution with profit at least $\max\{J_0, J_1\}$, which gives at least $1/2$ the profit of the optimal solution.

Due to lack of space, we omit the proof of the following lemma. It uses a modified version of the single-processor job scheduling algorithm of Bar-Noy et al. [1].

Lemma 6. *Given a set of jobs* $J_1, J_2, \ldots, J_k$, *and a path network* $(v_1, \ldots, v_n)$, *in polynomial time, we can compute a* 2*-approximation of the* best non-nesting *solution of the resource allocation problem.*

We can summarize the main result of this section in the following theorem.

Theorem 6. *Consider the resource allocation problem in the grid model for a* n*-node path topology. Suppose there are* k *jobs, each requiring bandwidth in the range* $[1, B]$. *Then, there is a polynomial time* $O(\log B)$*-approximation algorithm.*

Proof. We first partition all the requests into $\log B$ classes such that all jobs in one class have bandwidth requirement within a factor of two. When all bandwidth requests are in the range $[b, 2b]$ for some b, by Lemma 5, we can consider only non-nesting solutions at the expense of factor two in the approximation quality. For each of these $\log B$ classes of jobs, we run the approximation algorithm described in Lemma 6, which yields a factor 2-approximation of the best non-nesting solution. By choosing the best solution from the $\log B$ classes, we guarantee an approximation ratio of $O(\log B)$.

5 Related Work

Several grid systems have been developed, such as Globus [6], Legion [2], Condor [9] and SETI@Home [13], yet many interesting resource allocation problems in these systems remain to be addressed. Resource allocation schemes for grid computing include the market-based resource *sharing* as proposed by Chun and Culler [4], where *all* the jobs receive some resource, only the amount differs based on the offered price; the *SPAWN* model of Waldspurger et al. [11] essentially run parallel auctions for the different resources; the artificial economy model of Wolski et al. [12] uses supply and demand to set the prices. None of these models have any theoretical performance guarantees, or handle resource allocation with explicit bandwidth constraints.

Our resource allocation problem superficially resembles the multiple knapsack problem, but it differs considerably from the latter because in our problem jobs can be allocated across several different nodes if the bandwidth constraint is satisfied. Indeed, the multiple knapsack problem is a rather special case of the resource allocation problem (i.e. disjoint nodes topology).

For the special case of path topology, the resource allocation problem is similar to *Job Interval scheduling problem (JISP)*, where the input for each job is its length and a set of intervals, in which it can be scheduled. The objective is to maximize the number of scheduled jobs. JISP is strongly NP-Complete [8] and Chuzhoy et al. [5] gave a 1.582 approximation algorithm for it. Our model differs from JISP because there is no notion of profit associated with jobs in JISP. A more general version of JISP called *real time scheduling (RTP)* associates a weight with each job, and the objective is to maximize the total weight. BarNoy et al. [1] gave a 2-approximation algorithm for the the case of single machine. In section 4.2, we reduced the allocation problem for the path topology to RTP. This reduction however only works when there exist an optimal solution in which no link is used by more than one job, as RTP does not allow preemption. The scheduling techniques used in RTP can be applied to only path topologies as it is not at all clear how to reduce more general topologies to RTP.

6 Concluding Remarks

We studied an allocation problem motivated by grid computing and peer-to-peer systems. These systems pool together the resources of many workstations to create a virtual computing reservoir. Users can "draw" resources using a pay-as-you-go model, commonly used for utilities (electricity and water). As these technologies mature, and more advanced applications are implemented using computational grids, we expect providing bandwidth guarantees for the applications will become important. With that motivation, we studied the bandwidth-constrained allocation problems in grid computing.

Several open problems are suggested by this work. Is it possible to obtain a polynomial time $(1+\varepsilon)$-approximation scheme when $b_{\max} \leq B_{\min}$? If not, what is the best approximation factor one can achieve in polynomial time? In the global grid model, can one achieve a constant factor approximation independent of B? Extend our results to more general topologies than the path in the global grid model? Develop competitive algorithms for the online versions of the allocation problems.

References

1. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. In *Proc. of the 31st ACM Symp. on Theory of Computing*, pages 622 – 631, 1999.
2. S. Chapin, J. Karpovich, and A. Grimshaw. The legion resource management system. In *Workshop on Job Scheduling Strategies for Parallel Processing.*, 1999.
3. C. Chekuri and S. Khanna. A ptas for the multiple knapsack problem. In *Proc. of 11th Annual Symposium on Discrete Algorithms.*, pages 213–222, 2000.
4. B. Chun and D. E. Culler. Market-based proportional resource sharing clusters. Technical report, UC Berkeley, Computer Science, 2000.
5. J. Chuzhoy, R. Ostrovsky and Y. Rabani. Approximation algorithms for the job interval scheduling problem and realted scheduling problems. In *Proc. of 42nd Annual Symposium on Foundation of Computer Science.*, pages 348–356, 2001.
6. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Workshop on Job Scheduling Strategies for Parallel Processing.*, 1998.
7. I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. *2nd International Workshop on Peer-to-Peer Systems*, 2003.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.
9. M. Litzkow, M. Ivny, and M. Mutka. Condor—a hunter of idle workstations. In *Proc. of 8th International Conference on Distributed Computing.*, 1988
10. S. Martello and P. Toth. *Knapsack Problems—Algorithms and Computer Implementations*. John Wiley and Sons, New York, 1991..
11. C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and W. Stornetta. Spawn—a distributed computational economy. *IEEE Trans. on Software Engineering.*, 1992.
12. R. Wolski, J. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *Int. Journal of High Performance Computing Applications*, 2001.
13. http://setiathome.ssl.berkeley.edu. Seti@home.