

Application Placement on a Cluster of Servers (extended abstract) *

Bhuvan Urgaonkar, Arnold Rosenberg and Prashant Shenoy
Department of Computer Science,
University of Massachusetts, Amherst, MA 01003
{bhuvan, rsnbrg, shenoy}@cs.umass.edu

Abstract—The APPLICATION PLACEMENT PROBLEM (APP) arises in clusters of servers that are used for hosting large, distributed applications such as Internet services. Such clusters are referred to as hosting platforms. Hosting platforms imply a business relationship between the platform provider and the application providers: the latter pay the former for the resources on the platform. In return, the platform provider provides guarantees on resource availability to the applications. This implies that a platform should host only applications for which it has sufficient resources. The objective of the APP is to maximize the number of applications that can be hosted on the platform while satisfying their resource requirements. We show that the APP is NP-hard. Further, we show that even restricted versions of the APP may not admit polynomial-time approximation schemes. Finally, we present algorithms for the online version of the APP.

1 Introduction

Server clusters built using commodity hardware and software are an increasingly attractive alternative to traditional large multiprocessor servers for many applications, in part due to rapid advances in computing technologies and falling hardware prices. We call such server clusters *hosting platforms*. Hosting platforms can be shared or dedicated. In dedicated hosting platforms [1, 14], either the entire cluster runs a single application (such as a web search engine), or each individual processing element in the cluster is dedicated to a single application (such as dedicated web hosting services where each node runs a single application). In contrast, shared hosting platforms [3, 17] run a large number of different third-party applications (web-servers, streaming media servers, multi-player game servers, e-commerce applications, etc.), and the number of applications typically exceeds the number of nodes in the cluster. More specif-

ically, each application runs on a subset of the nodes and these subsets may overlap. Whereas dedicated hosting platforms are used for many niche applications that warrant their additional cost, economic reasons of space, power, cooling and cost make shared hosting platforms an attractive choice for many application hosting environments.

Shared hosting platforms imply a business relationship between the *platform provider* and the *application providers*: the latter pay the former for the resources on the platform. In return, the platform provider gives some kind of guarantees of resource availability to applications. This implies that a platform should admit only applications for which it has sufficient resources. In this work, we take the number of applications that a platform is able to host (admit) to be an indicator of the revenue that it generates from the hosted applications. The number of applications that a platform admits is related to the *application placement algorithm* used by the platform. A platform's application placement algorithm decides where on the cluster the different components of an application get placed. In this paper we study properties of the *application placement problem (APP)* whose goal is to maximize the number of applications that can be hosted on a platform. We show that APP is NP-hard and present approximation algorithms.

2 The Application Placement Problem

2.1 Notation and Definitions

Consider a cluster of n servers (also called nodes), N_1, N_2, \dots, N_n . Each node has a given *capacity* (of *available resources*). Unless otherwise noted, nodes are *homogeneous*, in the sense of having the same initial capacities. The APP appropriates portions of nodes' capacities; a node that still has its initial capacity is said to be *empty*. Let m denote the number of applications to be placed on the cluster and let us represent them as A_1, \dots, A_m . Further, each application is composed of one or more *capsules*. A capsule

*This research was supported in part by NSF grants CCR-9984030, EIA-0080119, CNS-0323597, CCF-0342417 and a gift from Intel Corporation.

may be thought of as the smallest component of an application for the purposes of placement — all the processes, data etc., belonging to a capsule must be placed on the same node. Capsules provide a useful abstraction for logically partitioning an application into sub-components and for exerting control over the distribution of these components onto different nodes. If an application wants certain components to be placed together on the same node (e.g., because they communicate a lot), then it could bundle them as one capsule. Some applications may want their capsules to be placed on different nodes. An important reason for doing this is to improve the availability of the application in the face of node failures — if a node hosting a capsule of the application fails, there would still be capsules on other nodes. An example of such an application is a replicated web server. We refer to this requirement as *the capsule placement restriction*. In what follows, we look at the APP both with and without the capsule placement restriction.

In general, each capsule in an application would require guarantees on access to multiple resources. In this work, we consider just one resource, such as the CPU or the network bandwidth. We assume a simple model where a capsule specifies its resource requirement as a fraction of the resource capacity of a node in the cluster (i.e., we assume that the resource requirement of each capsule is less than the capacity of a node). A capsule can be placed on a node only if the sum of its resource requirement and those of the capsules already placed on the node does not exceed the resource capacity of the node. We say that an application can be *placed* only if *all* of its capsules can be placed simultaneously. It is easy to see that there can be more than one way in which an application may be placed on a platform. We refer to the total number of applications that a placement algorithm could place as the *size* of the placement.

Lemma 1 *The APP is NP-hard.*

Proof: We reduce the well-known bin-packing problem [12] to the APP to show that it is NP-hard. We omit the proof here and present it in [16]. ■

2.2 Related Work

Two generalizations of the classical knapsack problem are relevant to our discussion of the APP. These are the *Multiple Knapsack Problem* (MKP) and the *Generalized Assignment Problem* (GAP). In MKP, we are given a set of n items and m bins (knapsacks) such that each item i has a profit $p(i)$ and a size $s(i)$, and each bin j has a capacity $c(j)$. The goal is to find a subset of items of maximum profit that has a feasible packing in the bins. MKP is a special case of GAP where the profit and the size of an item can vary based on the specific bin that it is assigned to. GAP is APX-hard

(see [12] for a definition of APX-hardness) and [15] provides a 2-approximation algorithm for it. This was the best result known for MKP until a polynomial-time PTAS was presented for it in [5]. It should be observed that the offline APP is a generalization of MKP where an item may have multiple components that need to be assigned to different bins (the profit associated with an item is 1). Further, [5] shows that slight generalizations of MKP are APX-hard. This provides reason to suspect that the APP may also be APX-hard (and hence may not have a PTAS).

Another closely related problem is a “multidimensional” version of the MKP where each item has requirements along multiple dimensions, each of which must be satisfied to successfully place it. The goal is to maximize the total profit yielded by the items that could be placed. A heuristic for solving this problem is described in [11]. However, the authors evaluate this heuristic only through simulations and do not provide any analytical results on its performance.

3 Hardness of Approximating the APP

In this section, we demonstrate that a restricted version of the APP does not admit a PTAS. The capsule placement restriction is assumed to hold throughout this section.

We give a gap-preserving reduction (see [16] for definition) from the *Multi-dimensional 0-1 Knapsack Problem* [2] to a restricted version of the APP.

Definition 1 Multi-Dimensional 0-1 Knapsack Problem (MDKP): *For a fixed positive integer k , the k -dimensional knapsack problem is the following:*

$$\text{Maximize } \sum_{i=1}^n c_i x_i$$

Subject to

$$\sum_{i=1}^n a_{ij} x_i \leq b_j, j = 1, \dots, k,$$

where: n is a positive integer; each $c_i \in \{0, 1\}$ and $\max_i c_i = 1$; the a_{ij} and b_i are non-negative real numbers; all $x_i \in \{0, 1\}$. Define $B = \min_i b_i$.

Hardness of approximating MDKP: For fixed k there is a PTAS for MDKP [10]. For large k the randomized rounding technique of [13] yields integral solutions of value $\Omega(OPT/d^{1/B})$. [4] shows that MDKP is hard to approximate within a factor of $\Omega(k^{\frac{1}{B+1}-\epsilon})$ for every fixed B , and establishes that randomized rounding essentially gives the best possible approximation guarantees.

Theorem 1 *Given any $\epsilon > 0$, it is NP-hard to approximate to within $(1 + \epsilon)$ the offline placement problem that has the*

following restrictions: (1) all the capsules have a positive requirement and (2) there exists a constant M , such that $\forall i, j (1 \leq j \leq k, 1 \leq i \leq n), M \geq b_j/a_{ji}$.

Proof: We explain later in this proof why the two restrictions mentioned above arise. We begin by describing the reduction.

The reduction: Consider the following mapping from instances of k -MDKP to offline APP:

Suppose the input to k -MDKP is a knapsack with capacity vector (b_1, \dots, b_k) . Also let there be n items I_1, \dots, I_n . Let the requirement vector for item I_j be (a_{j1}, \dots, a_{jk}) . We create an instance of offline APP as follows. The cluster has k nodes N_1, \dots, N_k . There are n applications A_1, \dots, A_n , one for each item in the input to k -MDKP. Each of these applications has k capsules. The k capsules of application A_i are denoted c_i^1, \dots, c_i^k . Also, we refer to c_i^j as the j^{th} capsule of application A_i . We now describe how we assign capacities to the nodes and requirements to the applications we have created. This part of the mapping proceeds in k stages. In stage s , we determine the capacity of node N_s and the requirements of the s^{th} capsule of all the applications. Next, we describe how these stages proceed.

Stage 1: Assigning capacity to the first node N_1 is straightforward. We assign it a capacity $C(N_1) = b_1$. The first capsule of application A_i is assigned a requirement $r_i^1 = a_{i1}$.

Stage s ($1 < s \leq k$): The assignments done by stage $s - 1$. We first determine the smallest of the requirements along dimension s of the items in the input to k -MDKP, that is, $r_{\min}^s = \min_{i=1}^n(a_{is})$. Next we determine the scaling factor for stage s , SF_s as follows:

$$SF_s = \lfloor C(N_{s-1})/r_{\min}^s \rfloor + 1. \quad (1)$$

Recall that we assume that $\forall s, r_{\min}^s > 0$. Now we are ready to do the assignments for stage s . Node N_s is assigned a capacity $C(N_s) = b_s \times SF_s$. The s^{th} capsule of application A_i is assigned a requirement $r_i^s = a_{is} \times SF_s$.

This concludes our mapping.

Correctness of the reduction: We show that the mapping described above is a reduction.

(\implies) Assume there is a packing P of size $m \leq n$. Denote the n items in the input to k -MDKP as I_1, \dots, I_n . Without loss of generality, assume that the m items in P are I_1, \dots, I_m . Therefore we have,

$$\sum_{i=1}^m a_{ij} \leq b_j, \quad j = 1, \dots, k. \quad (2)$$

Consider this way of placing the applications that the mapping constructs on the nodes N_1, \dots, N_k . If item $I_i \in P$, place application A_i as follows: $\forall j, 1 \leq j \leq k$, place capsule c_i^j on node N_j . We claim that we will be able to place

all m applications corresponding to the m items in P . To see why consider any node $N_i (1 \leq i \leq k)$. The capacity assigned to N_i is SF_i times the capacity along dimension i of the k -dimensional knapsack in the input to k -MDKP, where $SF_i \geq 1$. The requirements assigned to the i^{th} capsules of all the applications are also obtained by scaling by the same factor SF_i the sizes along the i^{th} dimension of the items. Multiplying both sides of (2) by SF_i we get,

$$SF_i \times \sum_{i=1}^m a_{ij} \leq SF_i \times b_j, \quad j = 1, \dots, k.$$

Observe that the term on the right is the capacity assigned to N_i . The term on the left is the sum of the requirements of the i^{th} capsules of the applications corresponding to the items in P . This shows that node N_i can accommodate the i^{th} capsules of the applications corresponding to the m items in P . This implies that there is a placement of size m .

(\Leftarrow) Assume that there is a placement L of size $m \leq n$. Let the n applications be denoted A_1, \dots, A_n . Without loss of generality, let the m applications in L be A_1, \dots, A_m . Also denote the set of the s^{th} capsules of the placed applications by $Cap_s, 1 \leq s \leq k$.

We make the following key observations:

- For any application to be successfully placed, its i^{th} capsule must be placed on node N_i . Due to the scaling by the factor computed in Eq. (1), the requirements assigned to the s^{th} ($s > 1$) capsules of the applications are strictly greater than the capacities of the nodes N_1, \dots, N_{s-1} . Consider the k^{th} capsules of the applications first. The only node these can be placed on is N_k . Since no two capsules of an application may be placed on the same node, this implies that the $k - 1^{\text{th}}$ capsules of the applications may be placed only on N_{k-1} . Proceeding in this manner, we find that the claim holds for all the capsules.
- Since for all s ($1 \leq s \leq k$), the node capacities and the requirements of the s^{th} capsules are scaled by the same multiplicative factor, the fact that the m capsules in Cap_s could be placed on N_s implies that the m items I_1, \dots, I_m can be packed in the knapsack in the s^{th} dimension.

Combining these two observations, we find that a packing of size m must exist.

Time and space complexity of the reduction: This reduction works in time polynomial in the size of the input. It involves k stages. Each stage involves computing a scaling factor (this involves performing a division) and multiplying $n + 1$ numbers (the capacity of the knapsack and the requirements of the n items along the relevant dimension).

Let us consider the size of the input to the offline placement problem produced by the reduction. Due to the scaling of capacities and requirements described in the reduction, the magnitudes of the inputs increase by a multiplicative factor of $O(M^j)$ for node N_j and the j^{th} capsules. If we assume binary representation this implies that the input size increases by a multiplicative factor of $O(M^{j/2})$, $1 < j \leq k$. Overall, the input size increases by a multiplicative factor of $O(M^k)$. For the mapping to be a reduction, we need this to be a constant. Therefore, our reduction works only when we impose the following restrictions on the offline APP: (1) k and M are constants, and (2) all the capsule requirements are positive.

Gap-preserving property of the reduction: The reduction presented is gap-preserving because the size of the optimal solution to the offline placement problem is *exactly equal* to the size of the optimal solution to MDKP.

$[\text{OPT}(\text{MDKP}) \geq 1] \implies [\text{OPT}(\text{offline APP}) \geq 1]$

$[\text{OPT}(\text{MDKP}) < 1] \implies [\text{OPT}(\text{offline APP}) < 1]$

Together, these results prove that the restricted version of the offline APP described in Theorem 1 may not admit a PTAS unless $P = NP$. ■

4 Offline Algorithms for APP

In this section we present approximation algorithms for several variants of the placement problem. Except in Section 4.4, we assume that the cluster is homogeneous, in the sense specified earlier. In most cases, we state the results without proof due to lack of space. We refer the reader to [16] for the proofs.

4.1 Placement of Single-Capsule Applications

We consider a restricted version of offline APP in which every application has exactly one capsule. We provide a polynomial-time algorithm for this restriction of offline APP, whose placements are within a factor 2 of optimal.

The approximation algorithm works as follows. Say that we are given n nodes N_1, \dots, N_n and m single-capsule applications C_1, \dots, C_m with requirements R_1, \dots, R_m . Assume that the nodes have unit capacities. The algorithm first sorts the applications in nondecreasing order of their requirements. Denote the sorted applications by c_1, \dots, c_m and their requirements by r_1, \dots, r_m . The algorithm considers the applications in this order. An application is placed on the “first” node where it can be accommodated (i.e., the node with the smallest index that has sufficient resources for it). The algorithm terminates once it has considered all the applications or it finds an application that cannot be placed, whichever occurs earlier. We call this algorithm FF_SINGLE.

Lemma 2 *FF_SINGLE has an approximation ratio of 2.*

4.2 Placement without the Capsule Placement Restriction

An approximation algorithm based on first-fit gives an approximation ratio of 2 for multi-capsule applications, provided that they don’t have the capsule placement restriction.

The approximation algorithm works as follows. Say that we are given n nodes N_1, \dots, N_n and m applications A_1, \dots, A_m with requirements R_1, \dots, R_m (the requirement of an application is the sum of the requirements of its capsules). Assume that the nodes have unit capacities. The algorithm first orders the applications in nondecreasing order of their requirements. Denote the ordered applications by a_1, \dots, a_m and their requirements by r_1, \dots, r_m . The algorithm considers the applications in this order. An application is placed on the “first” set of nodes where it can be accommodated (i.e., the nodes with the smallest indices that have sufficient resources for all its capsules). The algorithm terminates once it has considered all the applications or it finds an application that cannot be placed, whichever occurs first. We call this algorithm FF_MULTIPLE_RES.

Lemma 3 *FF_MULTIPLE_RES has an approximation ratio that approaches 2 as the number of nodes in the cluster grows.*

4.3 Placement of Identical Applications

Two applications are identical if their sets of capsules are identical. Below we present a placement algorithm based on “striping” applications across the nodes in the cluster and determine its approximation ratio.

Striping-based placement: Assume that the applications have k capsules each, with requirements r_1, \dots, r_k ($r_1 \leq \dots \leq r_k$). The algorithm works as follows. Let us denote the nodes as N_1, \dots, N_m . The nodes are divided into sets of size k each. Since $m \geq k$, there will be at least one such set. The number of such sets is $\lceil m/k \rceil$. Let $t = \lfloor m/k \rfloor, t \geq 1$. Let us denote these sets as S_1, \dots, S_{t+1} . Note that S_{t+1} may be an empty set, $0 \leq |S_{t+1}| \leq k - 1$. The algorithm considers these sets in turn and “stripes” as many unplaced applications on them as it can. The set of nodes under consideration is referred to as the *current set of k nodes*.

When the current set of k nodes gets exhausted and there are more applications to place, the algorithm takes the next set of k nodes and continues. The algorithm terminates when the nodes in S_t are exhausted, or all applications have been placed, whichever occurs earlier. Note that none of the nodes in the (possibly empty) set S_{t+1} are used for placing the applications.

Lemma 4 *The striping-based placement algorithm yields an approximation ratio of $\left(\frac{t+1}{t}\right)$ for identical applications, where $t = \lfloor m/k \rfloor$.*

4.4 Max-First Placement

In this section we turn our attention to the general offline APP. We let the nodes in the cluster be heterogeneous. We find that this problem is much harder to approximate than the restricted cases. We first present a heuristic that works differently from the first-fit based heuristics we have considered so far. We obtain an approximation ratio of k for this heuristic, where k is the maximum number of capsules in any application.

Our heuristic works as follows. It associates with each application a *weight* which is equal to the requirement of the *largest* capsule in the application. The heuristic considers the applications in nondecreasing order of their weights. We use a bipartite graph to model the problem of placing an application on the cluster. In this graph, we have one vertex for each capsule in the application and for each node in the cluster. Edges are added between a capsule and a node if the node has sufficient capacity for hosting the capsule. We say that the node is *feasible* for the capsule. In Lemma 5 (see [16] for the proof) we show that an application can be placed on the cluster if and only if there is a matching of size equal to the number of capsules in the application. We solve the maximum matching problem on this bipartite graph [7]. If the matching has size equal to the number of capsules, we place the capsules of the application on the nodes that the maximum matching connects them to. Otherwise, the application cannot be placed and the heuristic terminates. We refer to this heuristic as *Max-First*.

Lemma 5 *An application with k capsules can be placed on a cluster if and only if there is a matching of size k in the bipartite graph modeling its placement on the cluster.*

Lemma 6 *The placement heuristic Max-First described above has an approximation ratio of k , where k is the maximum number of capsules in an application.*

Proof: Let A represent the set of all the applications and $|A| = m$. Denote by n the number of nodes in the cluster and the nodes themselves by N_1, \dots, N_n . Let us denote by H the set of applications that Max-First places. Let O denote the set of applications placed by any optimal placement algorithm. Clearly, $|H| \leq |O| \leq m$. Represent by I the set of applications that both H and O place; that is, $I = H \cap O$. Further, denote by R the set of applications that neither H nor O places.

The basic idea behind this proof is as follows. We focus in turn on the applications that only Max-First and the optimal

algorithm place (that is, applications in $(H - I)$ and $(O - I)$), and compare the sizes of these sets. A relation between the sizes of these sets immediately yields a relation between the sizes of the sets H and O . (Observe that $(H - I)$ and $(O - I)$ may both be empty, in which case we have the claimed ratio trivially.)

Consider the placement given by Max-First. Remove from this all the applications in I , and deduct from the nodes the resources reserved for the capsules of these applications. Denote the resulting nodes by $N_1^{H-I}, \dots, N_n^{H-I}$. Do the same for the placement given by the optimal algorithm, and denote the resulting nodes by $N_1^{O-I}, \dots, N_n^{O-I}$. To understand the relation between the applications placed on these node sets by Max-First and the optimal algorithm, suppose Max-First places y applications from the set $(H - I)$ on the nodes $N_1^{H-I}, \dots, N_n^{H-I}$. Let us denote the applications in $(A - I)$ by $B_1, \dots, B_y, \dots, B_{|A-I|}$, where the applications are arranged in nondecreasing order of the size of their largest capsule. That is, $l(B_1) \leq \dots \leq l(B_y) \leq \dots \leq l(B_{|A-I|})$, $l(x)$ being the requirement of the largest capsule in application x . From the definition of Max-First, the y applications that it places are B_1, \dots, B_y . Also, the applications that the optimal algorithm places on the set of nodes $N_1^{O-I}, \dots, N_n^{O-I}$ must be from the set $B_{y+1}, \dots, B_{|A-I|}$. We make the following useful observation about the applications in the set $B_{y+1}, \dots, B_{|A-I|}$: *for each of these applications, the requirement of the largest capsule is at least $l(B_y)$* . Based on this we infer the following: Max-First will exhibit the worst approximation ratio when all the applications in $(H - I)$ have k capsules, each with requirement $l(B_y)$, and all the applications in $(O - I)$ have $(k - 1)$ capsules with requirement 0, and one capsule with requirement $l(B_y)$. Since the total capacities remaining on the node sets $N_1^{H-I}, \dots, N_n^{H-I}$ and $N_1^{O-I}, \dots, N_n^{O-I}$ are equal, this implies that in the worst case, the set $O - I$ would contain k times as many applications as $H - I$. Based on the above, we can prove an approximation ratio of k for Max-First as follows:

$$\begin{aligned} |O| &= |O - I| + |I| \leq k \cdot |H - I| + |I| \\ &\leq k \cdot (|H - I| + |I|) = k \cdot |H| \end{aligned}$$

This concludes our proof. \blacksquare

5 The Online APP

In the online version of the APP, the applications arrive one by one. We require the following from any online placement algorithm — *the algorithm must place a newly arriving application on the platform if it can find a placement for it without moving any already placed capsule*. This captures the placement algorithm's lack of knowledge of the requirements of the applications arriving in the future. We assume a heterogeneous cluster throughout this section.

5.1 Online Placement with Variable Preference for Nodes

In some scenarios, it may be useful to be able to honor any preference a capsule may have for one feasible node over another. In this section, we describe how online placement can take such preferences into account. We model such a scenario by enhancing the bipartite graph representing the placement of an application on the cluster by allowing the edges in the graph to have positive weights. The online placement problem therefore is to find the maximum matching of minimum weight in this weighted graph. We show that this can be found by reducing the placement problem to the *Minimum-weight Perfect Matching Problem*.

Our reduction works as follows. Assume that all the weights in the original bipartite graph are in the range (0, 1) and that they sum to 1. This can be achieved by normalizing all the weights by the sum of the weights. If an edge e_i had weight w_i , its new weight would be $\frac{w_i}{\sum_{e \in E} w_e}$. Denote the number of capsules by m and the number of nodes by n , $m \leq n$. Construct $n - m$ capsules and add edges with weight 1 each between them and *all* the nodes. We call these the *dummy capsules*.

Lemma 7 *In the weighted bipartite graph G corresponding to an application with m capsules and a cluster with n nodes ($m \leq n$), a matching of size m and cost c exists if and only if a perfect matching of cost $(c + n - m)$ exists in the graph G' produced by reduction described above.*

Proof: Due to lack of space we point the reader to [16] for the proof. ■

[9] gives a polynomial-time algorithm (called the *blossom* algorithm) for computing minimum-weight perfect matchings. [6] provides a survey of implementations of the blossom algorithm. The reduction described above, combined with Lemma 7, can be used to find the desired placement. If we do not find a perfect matching in the graph G' , we conclude that there is no placement for the application. Otherwise, the perfect matching minus the edges incident on the newly introduced capsules gives us the desired placement.

6 Conclusions

In this work we considered the offline and the online versions of APP, the problem of placing distributed applications on a cluster of servers. This problem was found to be NP-hard. We used a gap preserving reduction from the Multi-dimensional Knapsack Problem to show that a even a restricted version of the offline placement problem may not have a PTAS. A heuristic that considered applications in

nondecreasing order of their “largest component” was found to provide an approximation ratio of k , where k was the maximum number of capsules in any application. We also considered restricted versions of the offline APP in a homogeneous cluster. We found that heuristics based on “first-fit” or “striping” could provide an approximation ratio of 2 or better.

For the online placement problem, we allowed the capsules of an application to have variable preference for the nodes on the cluster and showed how a standard algorithm for the minimum weight perfect matching problem may be used to find the “most preferred” of all possible placements for such an application.

References

- [1] K. Appleby, S. Fakhouri, L. Fong, M. K. G. Goldszmidt, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano - SLA-based Management of a Computing Utility. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network Management*, May 2001.
- [2] A. K. Chandra, D. S. Hirschberg, and C. K. Wong. Approximate Algorithms for some Generalized Knapsack Problems. In *Theoretical Computer Science*, volume 3, pages 293–304, 1976.
- [3] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 103–116, October 2001.
- [4] C. Chekuri and S. Khanna. On Multi-dimensional Packing Problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 1999.
- [5] C. Chekuri and S. Khanna. A PTAS for the Multiple Knapsack Problem. In *Proceedings of the eleventh annual ACM-SIAM Symposium on Discrete algorithms*, 2000.
- [6] W. Cook and A. Rohe. Computing Minimum-weight Perfect Matchings. In *INFORMS Journal on Computing*, pages 138–148, 1999.
- [7] T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms. The MIT Press, Cambridge, MA.
- [8] D. S. Hochbaum (Ed.). Approximation Algorithms for NP-Hard Problems. PWS Publishing Company, Boston, MA.
- [9] J. Edmonds. Maximum Matching and a Polyhedron with 0,1 - Vertices. In *Journal of Research of the National Bureau of Standards 69B*, 1965.
- [10] A. M. Friese and M. R. B. Clarke. Approximation Algorithms for the m-dimensional 0-1 Knapsack Problem: Worst-case and Probabilistic Analyses. In *European Journal of Operational Research 15(1)*, 1984.
- [11] M. Moser, D. P. Jokanovic, and N. Shiratori. An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem. In *IEICE Trans. Fundamentals Vol. E80-A No. 3*, March 1997.
- [12] A Compendium of NP Optimization Problems. <http://www.nada.kth.se/viggo/problemist/compendium.html>.
- [13] P. Raghavan and C. D. Thompson. Randomized Rounding: a Technique for Provably Good Algorithms and Algorithmic Proofs. In *Combinatorica*, volume 7, pages 365–374, 1987.
- [14] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. QoS-Driven Server Migration for Internet Data Centers. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002)*, May 2002.
- [15] D. B. Shmoys and E. Tardos. An Approximation Algorithm for the Generalized Assignment Problem. In *Mathematical Programming A*, 62:461-74, 1993.
- [16] B. Urgaonkar, A. Rosenberg, and P. Shenoy. Application Placement on a Cluster of Servers. Technical Report TR04-18, Department of Computer Science, University of Massachusetts, March 2004.
- [17] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, December 2002.