



**The Association of System
Performance Professionals**

The **Computer Measurement Group**, commonly called **CMG**, is a not for profit, worldwide organization of data processing professionals committed to the measurement and management of computer systems. CMG members are primarily concerned with performance evaluation of existing systems to maximize performance (eg. response time, throughput, etc.) and with capacity management where planned enhancements to existing systems or the design of new systems are evaluated to find the necessary resources required to provide adequate performance at a reasonable cost.

This paper was originally published in the Proceedings of the Computer Measurement Group's 2007 International Conference.

For more information on CMG please visit <http://www.cmq.org>

Copyright 2007 by The Computer Measurement Group, Inc. All Rights Reserved

Published by The Computer Measurement Group, Inc., a non-profit Illinois membership corporation. Permission to reprint in whole or in any part may be granted for educational and scientific purposes upon written application to the Editor, CMG Headquarters, 151 Fries Mill Road, Suite 104, Turnersville, NJ 08012. Permission is hereby granted to CMG members to reproduce this publication in whole or in part solely for internal distribution with the member's organization provided the copyright notice above is set forth in full text on the title page of each item reproduced. The ideas and concepts set forth in this publication are solely those of the respective authors, and not of CMG, and CMG does not endorse, guarantee or otherwise certify any such ideas or concepts in any application or usage. Printed in the United States of America.

Improving Packing Algorithms for Server Consolidation

Yasuhiro Ajiro, Atsuhiko Tanaka
System Platforms Research Laboratories, NEC Corporation

Minimizing the number of servers and preparing sufficient resources are contradictory requirements in server consolidation. These requirements can be formalized as a variant of the bin packing problem, which is known to be NP-hard. The least-loaded, a load-balancing algorithm, is applied to the problem and compared to the classical First-Fit Decreasing algorithm designed to address the bin packing problem. Our technique of improving the algorithms is then presented.

1 Introduction

Server consolidation is classified into three stages—centralization, physical consolidation, and data and application integration [SER03]. Centralization involves moving servers to a common location. Physical consolidation involves moving a large number of source (existing) servers to a small number of high-performance target servers. Storage consolidation is also a kind of physical consolidation. Data and application integration involves integrating disparate data and applications into a common database and common application. The target of this research is physical consolidation serving as an engine of growth in the server consolidation market.

The growth of physical consolidation is due to server virtualization technology that enables multiple virtual machines, to which existing servers are moved, to *share* the physical resources of a computer. However, two contradictory requirements on shared resources must be satisfied at the same time. On the one hand, sufficient resources are required to avoid degradation in performance from the performance management point of view. Resource utilization is then a prime performance indicator because the sum of the utilizations of virtual machines running in a destination server must not exceed a threshold prescribed for that server. On the other hand, the number of destination servers has a significant impact on the cost of server consolidation. The number of destination servers is required to be as small as possible from the point of view of cost reduction.

This minimization can be formalized as a vector packing problem, which is a variant of the bin packing problem well known in the field of operations research. We are given items of different sizes in the bin packing problem and asked to pack them all into a minimum number of bins with a given capacity. Items for server consolidation are existing servers, item sizes are resource utilizations, bins are

destination servers, and the bin capacity is the utilization threshold of the destination servers. There are at least four kinds of utilizations for major physical resources, the CPU, disk, memory and network, of computers. The utilizations of these resources are summed and compared to thresholds independently of the other resources. The kinds of resource utilizations are dimensions in this vector packing problem.

Response time, which is also an important performance indicator, has to be taken into account for the network topology, bandwidth, and routers as well as for servers. Although response time is beyond the scope of this paper, that of servers is not expected to increase drastically in server consolidation. In terms of the M/M/1 queueing model, let ρ be the utilization, and $1/\mu$ be the service time, which is the minimum response time observed when a single request has been processed; then, the response time is expressed as $1/\mu(1 - \rho)$. The service times $1/\mu$ of most applications running efficiently on existing servers are sufficiently short and further reduced on the destination server whose performance may be several times higher than that of the existing servers. The response time cannot be more than a certain number of times longer than such a small $1/\mu$. For example, a response time is five times as long as a $1/\mu$ if $\rho = 0.8$ (80%).

The packing problem has a long history and a variety of applications in loading, layout design, scheduling, and material cutting. However, even the one-dimensional bin/vector packing problem is known to be NP-hard. NP-hard problems are problems that are as hard as the hardest problem in class NP. NP is the set of problems such that, when given a solution, whether it is a true(ly optimal) solution or not can be verified in polynomial time, i.e., $O(n^c)$ time, where n is the problem size (the number of items in the packing problem) and c is a constant. Naturally, finding an optimal solution needs more time, for example, exponential time $O(c^n)$, and is impossible in practice for not a small n . Even if $c = 2$ and $n = 100$, the exponential time will be almost 10^{30} . The “server” packing problem is also NP-

hard but is slightly different from the conventional packing problem in the sense that resource utilization often has a positive or negative correlation with some other resource utilization.

Engineers repeatedly make and remake server consolidation plans while grouping existing servers based on the corporate organization and network topology. Thus we need a better heuristic algorithm for finding a near-optimal solution to the server packing problem in reasonable time. Numerous algorithms have already been proposed for one- and two-dimensional bin packing problems, and First-Fit Decreasing (FFD) is one of the best [LMV02]. FFD and its family are greedy, i.e., items are packed as much as possible into currently prepared bins, and a new bin is added if an item cannot be packed into any of the current bins. Therefore, the FFD family unbalances the load between bins that are added early and late. This is why we compared FFD with the least loaded (LL), a load-balancing algorithm widely used in request-based systems. The load-balancing approach is more favorable for performance but has not yet been considered within the context of the packing problem.

Furthermore, we developed a new technique of improving the two algorithms. Our technique reduces the number of destination servers by re-ordering existing servers and retrying the packing procedure for either FFD or LL before a new server is added. The experiments we conducted with randomly generated data revealed that LL and our technique outperformed FFD. The remainder of this paper is organized as follows. We first formalize the server packing problem in Section 2. The algorithms and experiments are presented in Sections 3 and 4. Finally we conclude in Section 5.

2 Server Packing Problem

Dimensions in the server packing problem are resource utilizations, and a resource can have multiple utilizations. For example, disk utilization is referred to as the busy rate, the amount of disk usage, or the disk transfer per second, which further includes inbound and outbound transfers. Many such rates and amounts can be the dimensions, and the choice of utilizations depends on the measurement tools provided by the OS and the characteristics of the servers to be consolidated. Here, we have only considered two dimensions, the CPU and disk utilizations. Utilizations mean busy rates after this unless explicitly stated otherwise. We concentrated on the two utilizations to evaluate the fundamental consolidation efficiencies of algorithms with respect to the utilizations' correlations.

If two applications are running on the same server, the CPU utilization of the server is estimated as the sum of CPU utilizations accounted for by the two applications. This is the case with other resources and with the relation between existing and destination servers. For example, let (15%, 10%) be a pair of the CPU and disk utilizations of a server, and (20%, 12%) be that of another server. Then, the utilizations of a destination server accommodating the two servers are estimated at (35%, 22%), i.e., the sum of the vectors, regarding the pairs as two-dimensional vectors. In reality, utilizations are obtained as time-series data with the times of day and seasons. For simplicity, we considered the largest values in a time period here as the utilizations above. Time-series data can also be summed as a vector operation by aligning the data with the times of day. Server consolidation plans are made on the condition that the sum of utilizations for each resource do not exceed a threshold. Excess utilizations cause serious degradation in performance (recall the response time model $1/\mu(1-\rho)$ with utilization ρ being close to 1.0).

Here, we formalize the two-dimensional server packing problem. Let ρ_{c_i} and ρ_{d_i} be the CPU and disk utilizations of an existing server s_i ($i = 1, \dots, n$), X_j be a set of existing servers consolidated into a destination server s'_j ($j = 1, \dots, m$), and R_c and R_d be the thresholds of CPU and disk utilizations prescribed for destination servers. Thus n existing servers are all consolidated into m destination servers. The problem is then to minimize n under the constraints that $\sum_{s_i \in X_j} \rho_{c_i} \leq R_c$ and $\sum_{s_i \in X_j} \rho_{d_i} \leq R_d$.

There are yet two things to consider for utilizations and thresholds. First, if the CPU performance of a destination server is h times higher than that of an existing server, CPU utilization ρ measured on the existing server is converted into ρ/h on the destination server. We ignored this conversion by thinking of ρ_{c_i} above as a value that had already been converted. Second, the virtualization overhead increases the utilizations of servers running on virtual machines. We have to prepare proper margins of thresholds for the overhead.

Incidentally, items have to be packed without overlapping into bins for the bin packing problem as in Figure 1. If the sizes of the three items packed in the left bin represent CPU and disk utilizations, the sum of the utilizations is 1.9 and 1.3, which cannot be packed into a destination server with thresholds 1.0 and 1.0. Both bin and vector packing problems are the same problem only for one-dimensional cases. Dimensions are dependent for two- or more-dimensional bin packing problems, as seen in Figure 1. Because of such differences between the bin and vector packing problems, algorithms suited to the bin packing problem cannot always be used for the vector (server) packing problem.

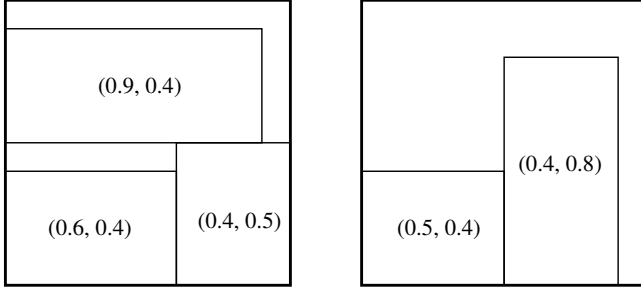


Figure 1: Instance of two-dimensional bin packing problem. Five items are packed into two bins. The pair of numbers on each item represents its width and height. The bin capacity is (1.0, 1.0).

```

sort existing servers to  $\{s_1, \dots, s_n\}$  in descending
order;
 $m \leftarrow 1$ ;  $X_1 \leftarrow \{\}$ ;
for  $i \leftarrow 1$  to  $n$  do
  for  $j \leftarrow 1$  to  $m$  do
    if packable( $X_j, s_i$ ) then
       $X_j \leftarrow X_j \cup \{s_i\}$ ;
      break
    fi
  end for;
  if  $j = m + 1$  then /* If fail to pack  $s_i$ , */
     $m \leftarrow m + 1$ ; /* a new server is added */
     $X_m \leftarrow \{s_i\}$  /* to have  $s_i$  */
  fi
end for

```

Figure 2: FFD algorithm.

3 Algorithms

3.1 First-Fit Decreasing

The FFD algorithm to address the server packing problem is shown in Figure 2. FFD receives n existing servers and sorts them in descending order of utilizations of a certain resource. The sorting is carried out for the largest (peak) utilizations within a time period even if time-series data are used. After the algorithm is executed, we obtain server accommodations X_j ($j = 1, \dots, m$), where m is the number of destination servers. The function $\text{packable}(X_j, s_i)$ returns true if packing existing server s_i into destination server s'_j satisfies the constraints (i.e., the utilization of s'_j does not exceed a threshold for any resource); otherwise it returns false.

FFD sequentially checks if all existing servers s_1, \dots, s_n can be packed into one of m current destination servers. FFD then packs s_i into a destination server first found to be able to accommodate it. If s_i cannot be packed into any current destination server, the $(m + 1)$ -th destination server is added and accommodates it. The complexity of this FFD algorithm is $O(n^2)$ because m is almost proportional to n . Here, we assumed the utilizations of no existing servers were beyond thresholds. Note that the binary search technique can reduce this complexity to $O(n \log n)$, but the sequential search is better for actual problems with time-series data. Even if server A is smaller than server B according to their peak utilizations in time-series data, and server C cannot be packed into A, then C can possibly be packed into B having a different peak time.

3.2 Least Loaded

The LL algorithm attempts to balance the load between servers by assigning incoming jobs to the least-loaded server. In server packing, we pack an existing server with a high utilization into a destination server with a low utilization. Figure 3 shows the LL algorithm we devised to address the server packing problem. The function $LB(\{s_1, \dots, s_n\})$ in the figure returns the theoretical lower bound for the number of destination servers that accommodate existing servers $\{s_1, \dots, s_n\}$. The lower bound is the smallest integer of numbers larger than the sum of the utilizations divided by a threshold. The lower bound for the CPU is $LB_c = \lceil \sum_{i=1}^n \rho_{c_i} / R_c \rceil$, while that for the disk is $LB_d = \lceil \sum_{i=1}^n \rho_{d_i} / R_d \rceil$. Function $LB(\{s_1, \dots, s_n\})$ returns the larger integer of the two lower bounds [CT01].

There are two differences between this LL and FFD. First, LL starts repacking after a new destination server is added when it has failed to pack an existing server into current m destination servers. This is aimed at balancing the load between a newly added destination server and the others. In contrast, FFD packs the existing server in question into a new destination server and continues to pack the remaining existing servers. LL initializes m to the lower bound to save time, even though we can also start with $m = 1$. Second, LL sorts destination servers (which accommodate X_1, \dots, X_m) in *ascending* order of utilizations each time before packing an existing server, so as to pack it into a less-loaded destination server.

The complexity of LL is $O(d \cdot n^2 \log n)$, where d is the difference between the lower bound and the final number m of destination servers. This complexity can be reduced to $O(d \cdot n^2)$ if we efficiently sort destination servers. The sorting does not actually require $O(n \log n)$ time but $O(n)$

```

sort existing servers to  $\{s_1, \dots, s_n\}$  in descending
order;
 $m \leftarrow LB(\{s_1, \dots, s_n\})$ ;
while true do
  for  $j \leftarrow 1$  to  $m$  do
     $X_j \leftarrow \{\}$  /* initialization */
  end for;
  for  $i \leftarrow 1$  to  $n$  do
    sort destination servers to  $\{X_1, \dots, X_m\}$ 
    in ascending order;
    for  $j \leftarrow 1$  to  $m$  do
      if packable( $X_j, s_i$ ) then
         $X_j \leftarrow X_j \cup \{s_i\}$ ;
        break
      fi
    end for;
    if  $j = m + 1$  then /* If fail to pack  $s_i$ , */
       $m \leftarrow m + 1$ ; /* a new server is added */
      break
    fi
  end for;
  if  $i = n + 1$  then /* all packed */
    break
  fi
end while

```

Figure 3: LL algorithm.

because only the utilizations of a destination server that has accommodated s_i is updated in iterations with i . The implementation we used in the experiments calls a built-in sorting function in JDK but might not cause any performance problems for a small n ($= 200$) as will be discussed.

3.3 Technique of Improving Algorithms

Both FFD and LL sort existing servers according to the utilizations of a certain resource. However, other resources may interfere with efficient packing and increase the number of destination servers. If sorting is done according to the CPU, an existing server with a low CPU utilization and a high utilization of another resource will be packed later than servers with high CPU utilizations. It is unlikely that there will be enough resources (except the CPU) left for the existing server when it is packed. We therefore developed the new algorithm shown in Figure 4 to solve this problem. In brief, it thinks of an existing server that has failed to be packed in currently prepared destination servers as one with a high utilization of a resource not considered

```

 $m \leftarrow LB(\{s_1, \dots, s_n\})$ ;
while true do
   $T' \leftarrow \{\}$ ;  $T \leftarrow \{s_1, \dots, s_n\}$ ;
  for  $r \leftarrow 1$  to MAXR do
    for  $j \leftarrow 1$  to  $m$  do
       $X_j \leftarrow \{\}$  /* initialization */
    end for;
     $s \leftarrow pack(T')$ ;
    if  $s \neq \emptyset$  then
      break
    fi;
     $s \leftarrow pack(T)$ ;
    if  $s \neq \emptyset$  then
       $T' \leftarrow T' \cup \{s\}$ ;
      continue
    else /* all packed */
      return  $m$ 
    fi
  end for;
   $m \leftarrow m + 1$ 
end while

```

Figure 4: Improved algorithm.

in sorting. The algorithm reduces the final number m of destination servers by packing such existing servers early.

The algorithm has two sets, T' and T , of existing servers; T' is packed prior to T . The function $pack(T)$ packs set T of existing servers into m destination servers similarly to the FFD or LL algorithm. The function differs from the two algorithms in that it removes existing server s , which has failed to be packed, from T and returns the s instead of adding a new destination server. To be more specific, the **if** statement of line 11 of FFD returns s_i if FFD is used as $pack()$. If LL is used as $pack()$, nothing but the sorting of the first line and the inside of the **for** loop of line 8 are necessary, and the **if** statement of line 17 returns s_i . Function $pack(T)$ returns an empty set \emptyset if all servers in T are successfully packed.

All existing servers initially belong to T . If $s (\neq \emptyset)$ is returned by $pack(T)$, the algorithm moves s to T' and retries packing T' and T from the beginning. The retries can be executed MAXR times at most unless T' and T are all packed. The algorithm otherwise terminates after returning m . The number of destination servers is incremented if the number of retries goes beyond MAXR or if packing T' fails. The MAXR is a constant given by the user, and its appropriate values will be considered in the next section.

The complexity of the improved algorithm is $O(d \cdot \text{MAXR} \cdot n^2)$ or $O(d \cdot \text{MAXR} \cdot n^2 \log n)$. The improved algorithm as well as FFD and LL can be applied to more than two-dimensional data, even though we used two-dimensional data in the experiments.

4 Comparison

We compared the FFD and LL algorithms and then evaluated our improved algorithm for randomly generated instances of the two-dimensional vector packing problem. The instances were a set of CPU and disk utilizations for 200 servers. We introduced the linear correlations of CPU and disk utilizations into the instances. With strong negative correlations, a resource not considered in sorting can strongly interfere with the packing done by FFD or LL. Our technique can contribute to improvements in these cases however. Conversely, assume that CPU and disk utilizations are equal for all servers. This is a special case of strong positive correlation. The instance is then simplified into a one-dimensional problem where FFD is assumed to work as intended.

Although we needed to generate random sequences of CPU and disk utilizations in the experiments that had several correlations, a standard method did not exist to our knowledge. We therefore used the probability P that both the CPU utilization and disk utilization of a server would be equal to or higher than the *reference* values, or that both utilizations would be lower than the reference values. We generated an instance of the CPU and disk utilizations of n existing servers with the algorithm:

```

for  $i \leftarrow 1$  to  $n$  do
   $\rho_{c_i} \leftarrow \text{rand}(2\bar{\rho}_c)$ ;  $\rho_{d_i} \leftarrow \text{rand}(\bar{\rho}_d)$ ;
   $r \leftarrow \text{rand}(1.0)$ ;
  if  $(r < P \wedge \rho_{c_i} \geq \bar{\rho}_c) \vee (r \geq P \wedge \rho_{c_i} < \bar{\rho}_c)$  then
     $\rho_{d_i} \leftarrow \rho_{d_i} + \bar{\rho}_d$ 
  fi
end for

```

where $\text{rand}(a)$ is a function that returns a random, uniformly distributed number of double type in the range $[0, a)$, which is 0 or larger and smaller than a ; $\bar{\rho}_c$ represents the reference CPU utilization, and $\bar{\rho}_d$ represents the reference disk utilization. We can control the correlations of CPU and disk utilizations to some extent by varying probability P .

We generated 100 instances each for two kinds of the refer-

ence values and five probabilities in the experiments, for a total of 1,000 instances. We set both $\bar{\rho}_c$ and $\bar{\rho}_d$ to 20% and then 40%. The distributions of CPU and disk utilizations range on $[0, 40\%)$ when $\bar{\rho}_c = \bar{\rho}_d = 20\%$, and $[0, 80\%)$ when $\bar{\rho}_c = \bar{\rho}_d = 40\%$. If one of the CPU and disk utilizations is apparently higher than the other for most servers, we can deal with instances as one-dimensional problems with respect to higher one. This is why we set both reference utilizations to the same value. We set the thresholds of both utilizations to $R_c = R_d = 80\%$ throughout the experiments. Although existing servers were sorted according to the CPU, which to use does not make a difference.

We set P to 0.00, 0.25, 0.50, 0.75, and 1.0 for $\bar{\rho}_c = \bar{\rho}_d = 20\%$, and then the average correlation coefficients became $-0.749, -0.380, -0.00534, 0.371,$ and 0.749 for each set of instances. These correlation coefficients correspond to strong-negative, weak-negative, no, weak-positive, and strong-positive correlations. The average lower bounds for the number of destination servers were 52.0, 51.7, 51.5, 51.1, and 51.1, which are all nearly a quarter of the 200 servers because on average 80/20 existing servers can be packed into a destination server. We similarly set P for $\bar{\rho}_c = \bar{\rho}_d = 40\%$. The correlation coefficients were then $-0.751, -0.375, -0.00190, 0.382,$ and 0.748 , and the lower bounds were 103, 103, 103, 103, and 102, which are nearly a half of 200.

The $\bar{\rho}_c$ is always the average CPU utilization of servers, and CPU utilizations are uniformly distributed on $[0, 2\bar{\rho}_c)$. Unlike $\bar{\rho}_c$, this is the case with the $\bar{\rho}_d$ and disk utilizations only if probability P is 0.0, 0.5, or 1.0. The average disk utilizations slightly decrease or increase if P is 0.25 or 0.75. Note that we compromised the uniformity of the distribution of disk utilizations for $P = 0.25$ and 0.75.

4.1 FFD vs. LL

Table 1 lists the average numbers m of destination servers obtained with the FFD and LL algorithms for each set of instances. The column ‘‘Corr’’ indicates the correlation coefficients for the CPU and disk utilizations of servers. The column ‘‘ m/LB ’’ indicates the ratios of m to the lower bounds LB and stands for consolidation efficiencies. The values m/LB closer to 1.00 mean higher efficiencies. The rightmost column indicates the average execution times for the algorithms. We implemented the algorithms in Java language (JDK 1.5) and executed them in Linux 2.4.33 on a machine equipped with a Pentium4, 2.0 GHz CPU and 768 MB of memory.

From Table 1 we can see that:

Table 1: Comparison of average numbers m of destination servers offered by FFD and LL for randomly generated instances.

$\bar{\rho}_c = \bar{\rho}_d = 20\%$				
Corr	Algorithm	m	m/LB	Time (sec)
-0.749	FFD	69.9	1.34	0.131
	LL	58.4	1.12	0.584
-0.380	FFD	65.2	1.26	0.128
	LL	57.2	1.11	0.496
-0.00534	FFD	61.4	1.19	0.125
	LL	56.1	1.09	0.462
0.371	FFD	58.1	1.14	0.121
	LL	55.5	1.09	0.396
0.749	FFD	55.0	1.08	0.118
	LL	53.5	1.05	0.253

$\bar{\rho}_c = \bar{\rho}_d = 40\%$				
Corr	Algorithm	m	m/LB	Time (sec)
-0.751	FFD	132	1.28	0.219
	LL	167	1.62	9.34
-0.375	FFD	132	1.28	0.219
	LL	160	1.55	8.21
0.00190	FFD	130	1.26	0.219
	LL	154	1.49	7.00
0.382	FFD	124	1.20	0.213
	LL	149	1.45	6.36
0.748	FFD	115	1.13	0.204
	LL	119	1.17	2.06

- For instances with lower correlation coefficients (i.e., with stronger negative correlations), the numbers m of destination servers are larger without respect to the algorithms or average (reference) utilizations. This supports our assumption that a resource not considered in sorting (disk in this case) interferes with packing.
- If $\bar{\rho}_c = \bar{\rho}_d = 20\%$, i.e., the average utilizations are low, LL always offers better efficiencies than FFD. The execution times for LL are also very short, even though they are longer than those of FFD.
- If $\bar{\rho}_c = \bar{\rho}_d = 40\%$, i.e., the average utilizations are high, FFD offers better efficiencies than LL, as expected. The efficiencies of LL degenerate considerably compared to the case with $\bar{\rho}_c = \bar{\rho}_d = 20\%$.

From the second observation, we can say that LL is more suitable for packing servers with low utilizations whose averages are a quarter of the thresholds or below. This was an unexpected result for us. The third observation demonstrates that FFD offers good efficiencies even in instances that LL cannot efficiently deal with.

4.2 Evaluation of Improved Algorithm

We investigated the numbers m of destination servers obtained with the improved algorithm for the same instances to evaluate the algorithm. We increased the value of $MAXR$ by 20 once at a time until the change in the number of destination servers decreased to less than one. We investigated both improved FFD and LL algorithms where we used FFD and LL as the function $pack()$ in Figure 4. The results are listed in Tables 2 and 3, where the column “Impr. Alg.” indicates the improved algorithms.

From the tables, we can see that:

- Both improved algorithms reduced the number m of destination servers. The rates of reduction were higher for lower correlation coefficients. This demonstrates that the improved algorithms can offer efficient packing even if a resource interferes with packing.
- The number m of destination servers decreases with increasing $MAXR$ and converges with some $MAXR$. The number m offered by the improved LL converges faster (i.e., with a smaller $MAXR$) than that by the improved FFD.
- If $\bar{\rho}_c = \bar{\rho}_d = 20\%$ (in Table 2), the improved LL offers better efficiencies than the improved FFD within shorter periods. The rates of reduction of m offered by the improved FFD are higher than those by the improved LL because the numbers m offered by FFD are larger as shown in Table 1. The improved FFD reduced m/LB from 1.08–1.34 to 1.06–1.11, while the improved LL reduced it from 1.05–1.12 to 1.03–1.07.
- If $\bar{\rho}_c = \bar{\rho}_d = 40\%$ (in Table 3), the convergence values of m are nearly equal without respect to the improved algorithms for each correlation coefficient. Since the numbers m offered by LL are much larger than those by FFD, the rates of reduction of m offered by the improved LL are considerably higher. The improved LL reduced m/LB from 1.17–1.62 to 1.09–1.19, while the improved FFD reduced it from 1.13–1.28 to 1.11–1.18.

These observations meant that our improved technique enabled LL to outperform FFD for any case in the experiments with respect to both consolidation efficiency and execution time.

Next, let us consider appropriate values for $MAXR$. From our experience, the CPU and disk utilizations of most (existing) servers have weak or even no correlation. To avoid one application from being affected by the failure of another, a server operates a single application in most cases.

Table 2: Numbers of destination servers offered by improved algorithms for instances with $\bar{\rho}_c = \bar{\rho}_d = 20\%$.

Corr: -0.749				
Impr. Alg.	MAXR	m	m/LB	Time (sec)
FFD	20	64.9	1.25	18.1
FFD	40	61.7	1.19	26.3
FFD	60	59.7	1.15	31.1
FFD	80	58.3	1.12	33.3
FFD	100	57.8	1.11	37.8
LL	20	55.8	1.07	3.92
LL	40	55.6	1.07	6.83

Corr: -0.380				
Impr. Alg.	MAXR	m	m/LB	Time (sec)
FFD	20	60.6	1.17	12.6
FFD	40	58.5	1.13	18.4
FFD	60	57.6	1.11	23.4
FFD	80	57.3	1.11	28.2
LL	20	54.9	1.06	3.35
LL	40	54.6	1.06	5.56

Corr: -0.00534				
Impr. Alg.	MAXR	m	m/LB	Time (sec)
FFD	20	57.9	1.12	9.49
FFD	40	56.8	1.10	14.8
FFD	60	56.5	1.10	20.6
LL	20	53.9	1.05	2.49
LL	40	53.7	1.04	4.16

Corr: 0.371				
Impr. Alg.	MAXR	m	m/LB	Time (sec)
FFD	20	55.8	1.09	7.14
FFD	40	55.5	1.09	11.7
LL	20	53.2	1.04	2.06
LL	40	53.1	1.04	3.46

Corr: 0.749				
Impr. Alg.	MAXR	m	m/LB	Time (sec)
FFD	20	54.1	1.06	4.38
FFD	40	54.1	1.06	7.31
LL	20	52.7	1.03	1.56
LL	40	52.6	1.03	2.70

Table 3: Numbers of destination servers offered by improved algorithms for instances with $\bar{\rho}_c = \bar{\rho}_d = 40\%$.

Corr: -0.751					
Impr. Alg.	MAXR	m	m/LB	Time (sec)	
FFD	20	126	1.22	48.0	
FFD	40	123	1.19	72.8	
FFD	60	122	1.18	100	
FFD	80	122	1.18	128	
LL	20	125	1.21	25.6	
LL	40	124	1.20	43.3	
LL	60	123	1.19	62.0	
LL	80	123	1.19	80.3	

Corr: -0.375					
Impr. Alg.	MAXR	m	m/LB	Time (sec)	
FFD	20	125	1.21	46.4	
FFD	40	122	1.18	68.6	
FFD	60	120	1.17	88.5	
FFD	80	120	1.17	112	
LL	20	122	1.18	21.3	
LL	40	121	1.17	36.6	
LL	60	121	1.17	52.5	

Corr: 0.00190					
Impr. Alg.	MAXR	m	m/LB	Time (sec)	
FFD	20	122	1.18	40.0	
FFD	40	119	1.16	55.6	
FFD	60	118	1.15	76.1	
LL	20	119	1.16	17.8	
LL	40	118	1.15	29.8	
LL	60	118	1.15	43.0	

Corr: 0.382					
Impr. Alg.	MAXR	m	m/LB	Time (sec)	
FFD	20	118	1.15	30.5	
FFD	40	117	1.14	48.4	
FFD	60	117	1.14	67.7	
LL	20	116	1.13	14.1	
LL	40	116	1.13	25.0	

Corr: 0.748					
Impr. Alg.	MAXR	m	m/LB	Time (sec)	
FFD	20	113	1.11	21.3	
FFD	40	113	1.11	37.2	
LL	20	111	1.09	9.58	
LL	40	111	1.09	17.1	

It is difficult for an application to fully utilize server resources. Not many servers use both resources evenly, and most servers heavily use either one. The numbers m offered by the improved FFD converge with a value of $MAXR$ of up to 60 for instances with weak or no correlations. The numbers m offered by the improved LL converge with a $MAXR$ of up to 40. For where $\bar{\rho}_c = \bar{\rho}_d = 40\%$ and where the correlation coefficient is -0.375 or 0.00190 , m decreases by only one if $MAXR$ is increased from 20 to 40. If we can do without this, 20 is sufficient as $MAXR$. After all, about 10% of existing servers (200 servers here) would be sufficient as $MAXR$ for the improved LL algorithm, while this would be 10–30% for the improved FFD. The average execution time for the improved LL is then at most 25.6 s (when $\text{Corr} = -0.751$ in Table 3), while that for the improved FFD is 100 s (in the same case) for packing 200 servers. Note that if the utilizations were not uniformly distributed, larger $MAXR$ might be necessary.

5 Conclusions

We applied LL, a load-balancing algorithm, to the server packing problem and compared it with the classical FFD algorithm. Furthermore, we developed improved FFD and LL algorithms and evaluated their consolidation efficiencies. The experiments we conducted revealed that LL was suitable for packing servers that had low utilizations but could not efficiently deal with servers that had high utilizations. The experiments further demonstrated that the improved LL outperformed the other algorithms and offered sufficient performance for packing 200 servers.

The essence of our improved algorithms was actually employed in our server consolidation planning tool, which could handle multiple resources and the time-series data of utilizations. The tool could also be used to estimate the necessary CPU performance (performance ratio to the CPU performances of existing servers) backward from a given number of destination servers.

An open question is how much worse the solutions offered by the improved algorithms are than exact solutions. Algorithms to find exact solutions have been studied in the literature [Spi94, CT01, CCM07]. Exact algorithms perform branch-and-bound searches with better lower bounds than the simple bound presented in this paper. These algorithms can find exact solutions for most instances with up to 100 items. Spieksma [Spi94] proposed a heuristic algorithm named FFD2. FFD2 sorts items according to $\lambda\rho_{c_i} + \rho_{d_i}$ and packs them to generate “well-filled” bins while varying the value of λ . Although FFD2 would be difficult to apply to more-dimensional problems and cause

load unbalance, it yielded promising results [CT01]. Comparing these algorithms with our improved algorithms is future work.

Coffman et al. [CGJ96] presented the average case analysis of the one-dimensional packing problem. Caprara and Toth [CT01] conducted experiments with randomly generated data, part of which had strong correlations, for the two-dimensional vector packing problem. We focused on the correlations and investigated the relationship between the correlations and consolidation efficiencies. Additionally, the characteristics of input data is sometimes beneficial to algorithms. For example, we could dynamically set an appropriate value for $MAXR$ in response to the correlation of utilizations.

Acknowledgments

We thank anonymous referees for their helpful comments. We learned from a referee that the Kepler conjecture, how best to pack identical spheres in three-dimensional Euclidean space, was stated in 1611 and proved recently.

References

- [CCM07] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional bin-packing problem with fixed orientation. *Operations Research Letters*, 35(3):357–364, 2007.
- [CGJ96] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS Publishing, 1996.
- [CT01] A. Caprara and P. Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111:231–262, 2001.
- [LMV02] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:379–396, 2002.
- [SER03] A. Spellmann, K. Erickson, and J. Reynolds. Server consolidation using performance modeling. *IT Professional*, 5:31–36, 2003.
- [Spi94] F. C. R. Spieksma. A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers and Operations Research*, 21(1):19–25, 1994.