Improving Packing Algorithms for Server Consolidation

YASUHIRO AJIRO, ATSUHIRO TANAKA

SYSTEM PLATFORMS RESEARCH LABORATORIES, NEC CORPORATION

PRESENTED BY : BASIL ALHAKAMI

Content

Introduction.

Background and Motivations.

Server Packing Problem.

Algorithms.

Improved Algorithms.

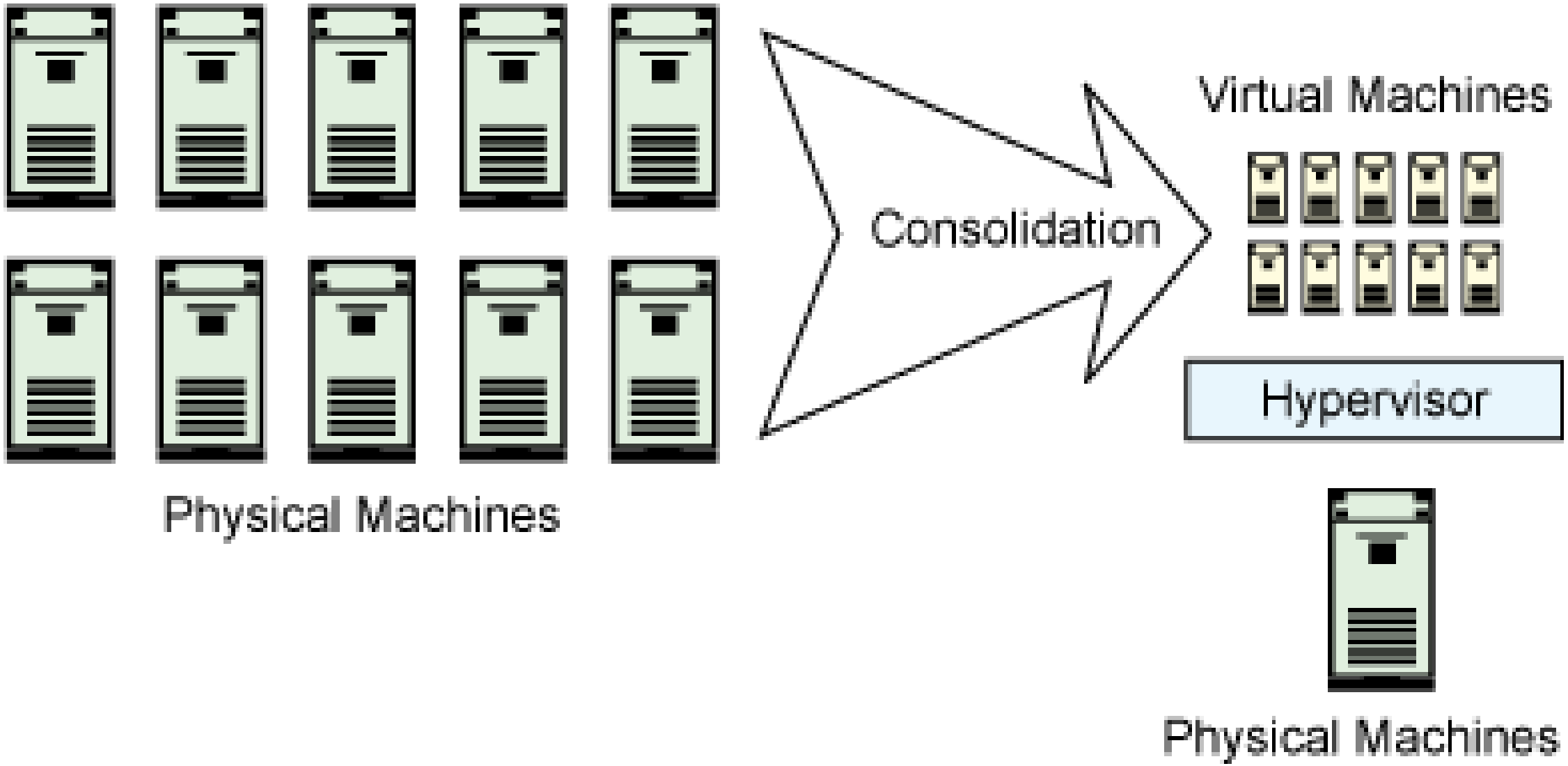
Comparison.

Conclusion.

Introduction

- Server Consolidation : An approach to the efficient usage of computer server resources in order to reduce the total number of servers or servers locations required

- The growth of server consolidation is due to virtualization which enables multiple virtual machines to share the physical resources of a computer. However, it should meet the following:
    1. Sufficient resources are needed to avoid degradation the performance : the utilization of the virtual machine must not exceed the threshold of that server.
    2. The numbers of destinations servers is required to be as small as possible.


Consolidation
Virtual Machines
Hypervisor
Physical Machines
Physical Machines

Background and Motivations

- Engineers repeatedly make server consolidation plans while grouping existing servers based on corporate organization and network topology.
- A better heuristic algorithm is needed to find a near-optimal solution to the server packing problem.
- First Fit Decreasing (FFD) is one of the best. However, FFD unbalances the load between bins that are added early and late.
- Least Loaded (LL) Algorithm is a Load balancing algorithm is more favorable for performance but not yet been considered within the context of packing problem.
- This paper introduce a new technique of improving both FFD and LL Algorithms.

Server Packing Problem (NP-hard)

A variant of the bin packing problem (well known in the field of operation research) :

- We are given a items of different sizes → current servers of different resource utilizations (CPU and disk utilization are considered in this research).
- We are asked to pack them into a minimum number of bins with a given capacity →destination servers and their utilization threshold.

Problem Formulation

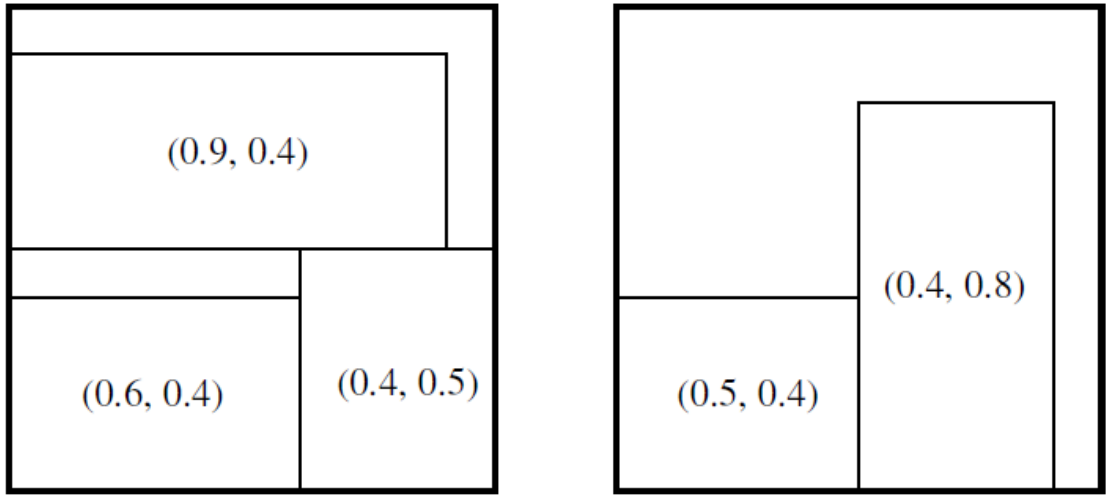
Dimension in server packing problem are resource utilization. In this research the focus is on CPU utilization and disk utilization :

Let ρ_{c_i} and ρ_{d_i} be the CPU and disk utilization of an existing server $s_i\ (i = 1, \cdots, n)$, X_i be a set of existing server consolidation into a destination server $s'_j\ (j = 1, \cdots, m)$, and R_c and R_d be the thresholds of CPU and disk utilization prescribed for the destination servers.

Thus n existing servers are all consolidated into m destination servers. The problem is then to minimize n under the constraint that $\sum_{s_i \in X_j} \rho_{c_i} \leq R_c$ and $\sum_{s_i \in X_j} \rho_{d_i} \leq R_d$.

Problem Formulation

- These items are packed into a bin with a capacity of (1.0,1.0).
- The sum of the utilization in the left bin is (1.9,1.3) which cannot be packed in a server with utilization of (1.0,1.0).



Algorithms

- First Fit Decreasing (FFD).
- Least Loaded (LL).

First Fit Decreasing (FFD)

FFD receives n existing servers and sort them in descending order of utilization of a certain recourse. The sorting is carried out for the largest utilization within a time period.

After the algorithm is executed we obtain server accommodations $X_j\ (j = 1, \cdots, m)$ where m is the number of destination servers.

The function *packable* (X_j, s_i) return true if packing existing server s_i into destination server s'_j satisfies the utilization of s'_j and does not exceed a threshold of any resource. Otherwise it return false.

First Fit Decreasing (FFD)

sort existing servers to $\{s_1, \cdots, s_n\}$ *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

Least Loaded (LL)

- The LL algorithm attempts to balance the load between servers by assigning incoming jobs to the least loaded servers.
- We pack an existing server with high utilization into a destination server with a low utilization.
- Authors devised LL algorithm to address the server packing problem.
- The function LB({s1…..sn)} returns the theoretical lower bound for the number of destination server that accommodate existing server {s1…..sn}.
- The lower bound is the smallest integer of numbers larger than the sum of the utilization divided by a threshold:

$$LB_c = \lceil \sum_{i=1}^{n} \rho_{c_i} / R_c \rceil \qquad LB_d = \lceil \sum_{i=1}^{n} \rho_{d_i} / R_d \rceil$$

The function LB({s1…..sn)} returns the larger integer of the two lower bounds.

Least Loaded (LL)

sort existing servers to $\{s_1, \cdots, s_n\}$ *in descending order*;
$m \leftarrow LB(\{s_1, \cdots, s_n\})$;
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 $\{X_1, \cdots, X_m\}$
    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

Differences between FFD and LL

- LL starts repacking after new destination server is added to balance the load between newly added destination and the others.
- FFD packs the existing servers into a new destination and continues to pack the remaining existing servers.
- LL sort destination server in ascending order of utilization each time before packing an existing server to pack into a less loaded destination server.

Improved Algorithm

- Both FFD and LL sort existing server according to the utilization of certain resource.
- Other resources may interfere with efficient packing and increase the number of destination servers.
- The improved algorithms address that issue.
- When a server fails to be packed it consider it as one with high utilization of resource not considered and pack such servers early.

Improved Algorithm

The algorithm has two sets of existing servers T' and T :

- T' is packed prior to T.
- The function *pack (T)* packs set T of existing servers into m destination servers.
- The difference is that the improved algorithm removes existing servers from T into T'.
- After packing all the severs in T the algorithm retries packing T' and T from the beginning.
- *MAXR* are the number of time the algorithm should retries at most unless T and T' are all packed.
- The number of destination servers incremented if the retries foes beyond *MAXR* or if packing T' fails.

Improved Algorithm

```
m ← LB({s_1, ··· , s_n});
while true do
  T' ← {};   T ← {s_1, ··· , s_n};
  for r ← 1 to MAXR do
    for  j ← 1 to m do
      X_j ← {}    /* initialization */
    end for;
    s ← pack(T');
    if s ≠ ∅ then
      break
    fi;
    s ← pack(T);
    if s ≠ ∅ then
      T' ← T' ∪ {s};
      continue
    else              /* all packed */
      return m
    fi
  end for;
  m ← m + 1
end while
```

Comparison

- The authors compared FFD and LL algorithms and then evaluated the improved algorithm for randomly generated instance of the two dimensional packing problem.
- The instances were a set of CPU and disk utilization for 200 servers.
- The linear correlation of CPU and disk utilization introduced into the instance.
- With strong negative correlation a resource not considered in the sorting can strongly interfere with the packing done by FFD and LL. The improved technique contribute to improvement in these cases.
- With strong positive correlation (CPU and disk utilization are equal) FFD is assumed to work as intended.

Comparison

FDD VS LL :

- With stronger negative correlation the number of destination servers are larger without respect to the average utilization.
- When the average utilization is low LL offers better efficiencies that FFD.
- When the average utilization is high FFD offers a much better efficiencies than LL.

$\overline{\rho_c} = \overline{\rho_d} = 20\%$

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

$\overline{\rho_c} = \overline{\rho_d} = 40\%$

–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

Improved Algorithms Comparison (20% average utilization)

- In this comparison the better MAXR can be also figured out.
- 10 % of the number of servers would be sufficient as MAXR for improved LL.
- 10-30% for improved FFD.
- If utilization were not uniformly distributed a larger MAXR might be necessary.

Impr. Alg.	*MAXR*	m	m/LB	Time (sec)
Corr: –0.749				
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				
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				
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

Improved Algorithms Comparison (40% average utilization)

- The average execution time for the improved LL is at most 25.6s when (Corr= -0.751).
- The average execution time for the improved FFD 100s (Corr= -0.751).

Impr. Alg.	*MAXR*	m	m/LB	Time (sec)
Corr: –0.751				
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				
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				
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

Conclusion

- A Load balancing algorithm were applied to the server packing problem in this research.
- LL were compared to FFD and revealed that LL was suitable only for packing servers that had low utilization.
- The improved LL outperformed the other algorithms and offered sufficient performance for packing 200 servers.

Thank You