Journal of Computer Science 5 (12): 956-961, 2009
ISSN 1549-3636
© 2009 Science Publications

Server Optimization Using Heuristic Algorithms for Dynamic-Split-and-Merge Scheme in Wireless Multicast

[1]R. Sukumar and [2]V. Vasudevan
[1]Department of Computer Science and Engineering,
Sethu Institute of Technology, Kariapatti, Virudhunagar District, Tamilnadu, India
[2]Department of Information Technology,
Arulmigu Kalasalingam College of Engineering, Krishnankovil, India

Abstract: Problem statement: In order to minimize the overall network traffic in a multiserver system, the number of users served by each server (and hence the group size) should remain constant. As the underlying traffic fluctuates, a split and merge scheme is implemented in a physical server to achieve load balancing. **Approach:** Minimizing the number of servers during the merge operation is NP hard and to achieve these two algorithms namely FFD bin packing algorithm and LL algorithm were proposed to find the near optimal values of destination servers. **Results:** The performance of these algorithms were analyzed and compared based on several parameters. **Conclusion:** Results showed that LL algorithm outperforms FFD algorithm.

Key words: Heuristic algorithm, load balancing, dynamic split and merge, destination servers

INTRODUCTION

The number of users in a multicast group tends to fluctuate due to frequent user join/leave. In order to handle key management efficiently and reduce the join/leave latency a dynamic split and merge scheme is suggested[5,7]. If the number of users in a server is greater than $\varnothing_{max}$, the server is split into several logical servers for which the number of users in each server is as close as possible to the optimal group size ρ/m^* . If there are some servers in which the total number of users is less than $\varnothing_{min}$, the groups are merged into a single logical server with the goal of getting as close as possible to ρ/m^*. The problem of finding proper groups to be merged is NP-hard. NP is the set of problems such that, when given a solution, whether it is a truly 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[12]. 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" merging problem is also NP hard and the number of destination servers is required to be as small as possible from the point of view of cost reduction and manageability. This minimization can be formalized as a bin packing problem well known in the field of operations research[8]. We are given items of different sizes in the bin packing problem and asked to pack them all into a minimum n umber of bins with a given capacity. Items for server consolidation are existing servers, item sizes are group sizes of different servers and bins are destination servers.

An important parameter to study the performance of server packing algorithms is the server response time. For a server packing algorithm to exhibit good convergence, response time is not expected to increase drastically. For example in a M/M/1 queuing 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 time $1/\mu$ 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 $1/\mu$ if $\rho = 0.8$ (80%).

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

Corresponding Author: R. Sukumar, Department of Computer Science and Engineering, Sethu Institute of Technology, Pulloor, Kariapatti-626 106. Virudhunagar District, Tamil Nadu, India
Tel: 04566-308001 Fax: 04566-308000

packing problems and First-Fit Decreasing (FFD) is one of the best[9]. FFD and its family are greedy, i.e., items are packed as much as possible into currently prepared bins and new bin 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[13]. 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.

MATERIALS AND METHODS

Related works: Much of the previous research on server optimization has been done without considering the dynamic nature of the multicast group members. This body of work includes dynamic split and merge scheme for large scale wireless multicast. Present research is based on the scheme given in[6,7] and we model and analyze it. Previous research address mainly reducing number of existing servers and has considered neither a dynamic split and merge scheme nor the comparison between FFD and LL algorithms.

Teo[3] focuses on an experimental analysis of the performance and scalability of cluster-based web servers. The three dispatcher-based scheduling algorithms analyzed are: Round robin scheduling, least connected based scheduling and least loaded based scheduling. The least loaded algorithm is used as the baseline (upper performance bound) in the analysis and the performance metrics include average waiting time, average response time and average web server utilization. It is found that the least connected algorithm performs well for medium to high workload.

Shen *et al*.[4] present heuristic algorithms that may be used for light-path routing and wavelength assignment in optical WDM networks under dynamically varying traffic conditions. They considered both the situations where the wavelength continuity constraint is enforced or not enforced along a light-path. The performance of these algorithms has been studied through simulations. A comparative study on their performance with that of a simpler system that uses fixed shortest-path routing has been performed. The proposed algorithms provided lower blocking probabilities and are simple enough to be applied for real time network control and management. They have also studied that the heuristic algorithms are computationally simple and efficient to implement and provide good wavelength utilization leading to efficient usage of the network's resources.

Türkay Dereli and Sena Daş[15] studied a hybrid Simulated-Annealing (SA) algorithm for the 2-Dimensional (2D) packing problem. A recursive procedure has been used in the proposed algorithm to allocate a set of items to a single object. The problem has been handled as a permutation problem and the proposed recursive algorithm is hybridized with the simulated annealing algorithm. The effectiveness of the algorithm has been tested on a set of benchmark problems. The computational results have shown that the algorithm gives promising results.

Zhao and Yang[2] proposed an accumulated k-subset algorithm (AK algorithm) to balance load in distributed SLEE. Based on a model of resource heterogeneity and load vector, they have found that the AK algorithm improves the k-subset algorithm by accumulating load information within every update interval. Experiments on different update intervals and request arrival rates suggested AK further reduces herd effect due to stale load information and outperforms k-subset algorithm by 5-10%. F. Clautiaux *et al*.[5] proposed a new exact method for the well-known two-dimensional bin-packing problem. It is based on an iterative decomposition of the set of items into two disjoint subsets. They have tested the efficiency of this method against benchmarks of the literature.

Dynamic split and merge: Since the number of users in a multicast group tends to fluctuate, the system can have variable number of servers. During a busy period when more number of users join the group, number of servers can be more and during a quiet period, the number of servers can be less in order to handle the key management efficiently. We therefore fix a threshold n_{max}, for the maximum number of users in a group and $\varnothing_{max}$, for maximum number of servers the system can have at a particular period of time. This is due to the fact that more number of servers adds to the complexity of the system.

The number of servers the system needs at a particular period of time is decided by the following procedure:

- Step 1: Fix a threshold for n_{max} and n_{min}
- Step 2: If $n > n_{max}$, Split the group
- Step 3: If $n < n_{min}$, Merge the group

Merging a group with some other group is done in such a way that the total number of users in the merged group does not exceed n_{max}. Therefore, before merging a group we must find the possible groups that can be merged.

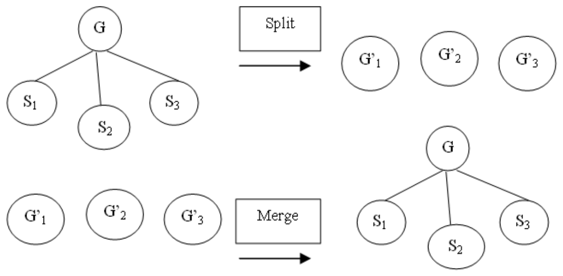

Fig. 1: Splitting and merging for K = 3

Where, n_{max} and n_{min} represent maximum and minimum number of users in a group respectively.

Initially there will be a single server and when more number of users join the group multiple servers are introduced into the system. We use the LKH for generation and distribution of group keys.

We fix a threshold for number of users in a group and when the number of users goes beyond this value we dynamically split the servers. In the same way, when the number of users fall below the threshold value we merge the servers.

Figure 1 shows an example of merging and splitting for K = 3. If there is a group in which the total number of users, n, is greater than max_t, the group is split into three sub groups and the original subgroup keys, S_1, S_2 and S_3 become the new group keys, G'_1, G'_2 and G'_3, for these three new groups respectively. Whereas, if there are three groups in which n is less than min_t, the groups are merged and generate a new group key is generated. The original group keys, G'_1, G'_2 and G'_3, become subgroup keys, S_1, S_2 and S_3, which can be used to encrypt the new group key, G that is sent to these three groups. Hence, the new merged group will have three sets of message overhead, one for each subgroup.

In order to tackle this problem several algorithms have been proposed in the bin packing context for consolidating items into minimum number of bins. In this study First-Fit Decreasing (FFD) bin-packing algorithm and the Least Loaded (LL) are used[14]. Both these algorithms are given the same input and the results are compared for various number of servers. Two parameters are considered for comparison: The time complexity and the number of destination servers.

Algorithms: Two heuristic algorithms, FFD bin packing algorithm and LL algorithm that are evaluated in our experiments are discussed below. We study the performance of FFD bin packing algorithm and the LL algorithm. These algorithms were chosen because they are some of the mostly used algorithms in this field and are fairly simple to implement and do not add redundant delays in the system.

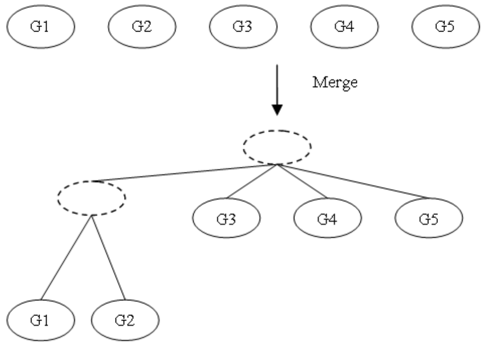

Fig. 2: An example of server merge operation

First-fit decreasing bin packing algorithm: In the FFD algorithm, items are first sorted in decreasing order of size[6]. The FFD algorithm to address the server packing problem is shown in Fig. 2. There are a number of empty bins of size with increasing index. The items are placed into the bins one by one, placing each item in the first bin in which it will fit (i.e., the total size of items in the bin does not exceed) in a round-robin manner. The time complexity of FFD algorithm is shown to be O(n log n), where n is the number of items.

FFD algorithm is applied for merging servers. Each server is considered as an item with its group size as the item size. Assuming that there are many bins with size of $\varnothing_{min}$, packing operation is done in such a way that, the number of nonempty bins is very close to the optimal number of servers. Therefore, each bin should be filled as much as possible. After packing the groups into the bins, the groups can be merged in a bin into a new larger group served by a single logical server.

The following example demonstrates a simple method to merge the trees. In order to keep the new key tree as short and as balanced as possible, the taller trees (i.e., a tree with greater depth) are added into higher level nearer to the root (i.e., level) while the shorter ones into the lower level (i.e., level). Figure 2 shows a case of merging five servers with a branching factor of 4. If G1 and G2 are the shortest two trees, these two trees are added into the second level and the taller trees are added into the first level. The dotted ovals represent the new nodes created after merging.

The FFD algorithm to address the server packing problem is shown in Fig. 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[10]. After the algorithm is executed, we obtain server accommodations X_j(j = 1,....,m), where m is the number of destination servers. The function 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[8,15].

FFD sequentially checks if all existing servers $s_1,....,s_n$ 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 algorithms 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.

Least loaded algorithm: The LL algorithm works on the principle of load balancing. The LL algorithm attempts to balance the load between servers by assigning incoming jobs to the least-loaded server[1,12]. In server packing, an existing server with a high utilization is packed into a destination server with a low utilization[11]. Figure 3 shows the LL algorithm that addresses the server packing problem. The function LB $(\{s_1,.....s_n\})$ in Fig. 3 returns the theoretical lower bound for the number of destination servers that accommodate existing servers $\{s_1,.....s_n\}$. 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 $(\{s_1,.....s_n\})$ returns the larger integer of the two lower bounds [CT01].

There are two differences between 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 $X_1,.....X_m$) in ascending order of utilizations each time before packing an existing server, so as to pack it into a less-loaded destination server

```
Sort existing servers to {s1,...,sn} in descending order;
m ← 1; X1 ← {};
for i← 1 to n do
        for j← 1 to m do
                if packable(Xi, si) then
                        Xj ← Xj ∪ {si};
                        break
                fi
        end for;
        if j = m+1 then        /* If fail to pack si */
                m ← m+1;       /* a new server is added */
                Xm ← {si}      /* to have si */
        fi
end for
```

Fig. 3: FFD algorithm

```
sort existing servers to {s1,.....sn} in descending order;
m ← LB ({S1,....Sn});
while true do
for j ← 1 to m do
Xj ← {} /* initialization *?
end for;
for i ← 1 to n do
sort destination servers to {X1,...Xm} in ascending order;
for j ← 1 to m do
if packable (Xj, Si) then Xj ← Xj ∪{Si};
break
fi
end for;
if j = m + 1 then /* If fail to pack si, */
m ← m +1;    /* a new server is added */
break
fi
end for;
if i = n + 1 then /* all packed */
break
fi
end while
```

Fig. 4: LL algorithm

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) because only the utilizations of a destination server that has accommodated s_i is updated in iterations with i.

RESULTS AND DISCUSSION

Table 1 shows the average numbers m of destination servers obtained with the FFD and LL algorithms for each n value. The column "$m\ LB^{-1}$" indicates the ratios of m to the lower bounds LB and stands for consolidation efficiencies. The values $m\ LB^{-1}$

closer to 1.00 mean higher efficiencies. The rightmost column indicates the average execution times for the algorithms. The algorithms have been implemented in java language (JDK 1.5). Figure 5 shows the comparison between FFD and LL algorithm based on number of destination servers. The results show that while m increases linearly with n, LL algorithm results in the better m values compared to FFD algorithm. Figure 6 shows the comparison between FFD and LL algorithm based on lower bound for the number of destination servers that accommodate existing servers $\{s_1,.....s_n\}$.

Table 1: Comparison of average number m of destination servers offered by FFD and LL for various n values

n	Algorithm	m	m LB^{-1}	Time (sec)
50	FFD	39.6	1.34	0.061
	LL	37.0	1.12	0.073
100	FFD	87.3	1.26	0.069
	LL	84.2	1.11	0.078
150	FFD	131.7	1.19	0.082
	LL	127.0	1.09	0.188
200	FFD	188.0	1.14	0.127
	LL	171.0	1.09	0.284
250	FFD	217.0	1.08	0.142
	LL	203.0	1.05	0.323

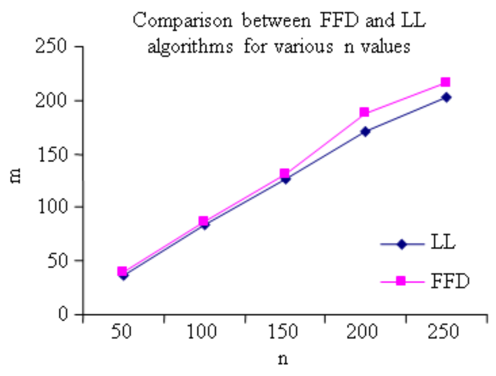

Fig. 5: Comparison of FFD and LL based on m

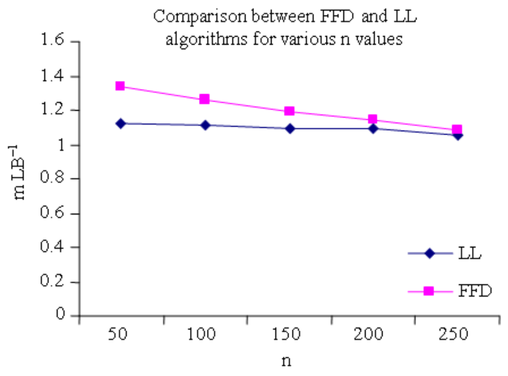

Fig. 6: Comparison of FFD and LL based on m LB^{-1}

It is understood that for smaller values of n there is a moderate difference in the performance of the LL algorithm compared to FFD algorithm. Figure 7 shows the comparison between FFD and LL algorithm based on execution time. It is clear that the execution time for LL algorithm for larger values of n is very high compared to FFD algorithm.

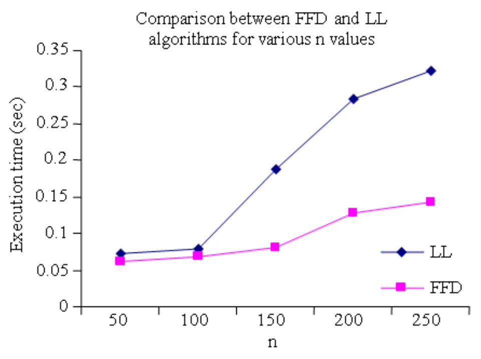

Fig. 7: Comparison of FFD and LL based on convergence time

CONCLUSION

In order to efficiently handle the frequent membership change in a multicast system, a dynamic split and merge technique has been proposed. Two algorithms, FFD and LL, have been suggested to get near optimal values for number of destination servers during the merge operation. Comparison between FFD and LL algorithm shows that the convergence time is lower for FFD, whereas LL algorithm performs well in getting the number of destination servers very close to the optimal value and balances the load better than FFD.

REFERENCES

1. Cardellini, V., M. Colajanni and P.S. Yu, 1999. Redirection algorithms for load sharing in distributed web-server systems. Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, May 31-June 4, IEEE Computer Society, Washington DC., USA., pp: 528. http://portal.acm.org/citation.cfm?id=880581&dl=&coll=
2. Zhao, Y. and F. Yang, 2006. A dynamic load balancing algorithm for distributed SLEE in mobile service provisioning. Proceeding of the International Conference on Wireless Communications, Networking and Mobile Computing, Sept. 22-24, IEEE Computer Society, Washington DC., USA., pp: 1-4. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4149561

3. Teo, Y.M., 2001. Comparison of load balancing strategies on cluster-based web servers. Simulation, 77: 185-195. http://sim.sagepub.com/cgi/content/abstract/77/5-6/185
4. Shen, G., S.K. Bose, T.H. Cheng, C. Lu and T.Y. Chai, 2001. Efficient heuristic algorithms for light-path routing and wavelength assignment in WDM networks under dynamically varying loads. Comput. Commun., 24: 364-373. DOI: 10.1016/S0140-3664(00)00236-X
5. Clautiaux, F., J. Carlier and A. Moukrim, 2007. A new exact method for the two-dimensional bin packing problem with fixed orientation. Operat. Res. Lett., 35: 357-364. DOI: 10.1016/j.orl.2006.06.007
6. Caprara, A. and P. Toth, 2001. Lower bounds and algorithms for two dimensional vector packing problem. Discrete Applied Math., 111: 231-262. http://portal.acm.org/citation.cfm?id=508254
7. Lodi, A., S. Martello and D. Vigo, 2002. Recent advances on two dimensional bin packing problem. Discrete Applied Math., 123: 379-396. DOI: 10.1016/S0166-218X(01)00347-X
8. Spellmann, K.E. and J. Reynolds, 2003. Server consolidation using performance modeling. IT Professional, 5: 31-36. DOI: 10.1109/MITP.2003.1235607
9. Eager, D.L., E.D. Lazowska and J. Zahorjan, 1986. A comparison of receiver-initiated and sender-initiated adaptive load sharing. Perform. Evaluat., 6: 53-68. DOI: 10.1016/0166-5316(86)90008-8
10. Goldberg, A.P., G.J. Popek and S.S. Lavenberg, 1983. A validated distributed system performance model. Proceedings of the 9th International Symposium on Computer Performance Modeling, Measurement and Evaluation, May 25-27, North-Holland Publishing Co., Amsterdam, The Netherlands, pp: 251-268. http://portal.acm.org/citation.cfm?id=724593
11. Hać, 1986. A distributed algorithm for performance improvement through replication and migration. Proceeding of the IEEE Computer Networking Symposium, Nov. 17-18, Washington, DC., USA., pp: 163-168. http://portal.acm.org/citation.cfm?id=76153
12. Tantawi, A.N. and D. Towsley, 1985. Optimal static load balancing in distributed computer systems. J. ACM., 32: 445-465. http://portal.acm.org/citation.cfm?doid=3149.3156
13. Wang, Y.T. and R.J.T. Morris, 1985. Load sharing in distributed systems. IEEE Trans. Comput., C-34: 204-217. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1676564
14. Baker, B.S., 1985. A new proof for the first-fit decreasing bin-packing algorithm. J. Algorithms, 6: 49-70. http://cat.inist.fr/?aModele=afficheN&cpsidt=9264746
15. Türkay dereli and G. Sena Daş, 2002. A hybrid simulated annealing algorithm For 2d packing problems. Proc. Int. Symp. Intell. Manufactur. Syst., 1: 959-966.