Using H-Algorithm to Find the Study of Multi-Server Wireless Multicast System

Dr.A.Arul Lawrence selvakumar[1], N. K.Prema[2]
Director / Department of CA, Adhiparasakthi Engineering College[1],
Asst.Professor/Department of CSE, IFET College of Engineering[2],
Anna Technical University, Chennai, INDIA, [1, 2]
Aarul72@hotmail.com[1], premasenthi@gmail.com[2]

Abstract: *In order to minimize the overall network traffic in a multi-server wireless multicast 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. 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 are proposed to find the near optimal values of destination servers. The performance of these algorithms are analyzed and compared based on several parameters. Results show that LL algorithm outperforms FFD algorithm.*

Keywords: heuristic algorithm, load balancing, dynamic split and merge, destination servers, response time.

1. 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 $Ø_{max}$, 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 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 δ 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 $1/\mu(1-\delta)$. 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 $\delta = 0.8$ (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 packing problems and First-Fit Decreasing (FFD) is one of the best. 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 group size ρ/g. If there are some servers in which the total number of users is less than $Ø_{min}$, the groups are merged into a single logical server with the goal of getting as close as possible to ρ/g. 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 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" 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. 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 group 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. The rest of the paper is organized as follows. Section 2 outlines some of the related work in group key management. Section 3 describes a dynamic merge and split scheme. The detailed explanation on FFD and LL algorithms are given in section 4. The results of the analysis and discussion are given in Section 5. Concluding remarks are provided in section 6.

2 RELATED WORKS

Much of the previous work 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. Our work is based on the scheme given in [6] and [7], and we model and analyze it. Previous works 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. Yong Meng Teo (2001) 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.

G. Shen et al (2001) 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 G. Sena Daş (2002) studied a hybrid simulated-annealing (SA) algorithm for the two-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.

Yao Zhao and Fangchun Yang (2006) 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 (2007) 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.

3. DYNAMIC SPLIT AND MERGE SCHEME

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

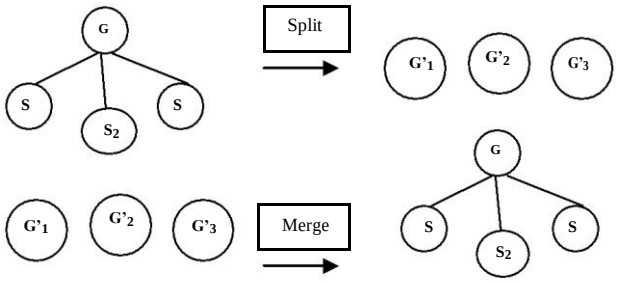

Figure 1: Splitting and Merging for K=3

the key management efficiently. We therefore fix a threshold $Ø_{max}$, for the maximum number of users in a group and $Ø_{min}$, for minimum number of users a server 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 $Ø_{max}$ and $Ø_{min}$
Step 2: If u > $Ø_{max}$, Split the group
Step 3: If u < $Ø_{min}$, 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 $Ø_{max}$. Therefore, before merging a group we must find the possible groups that can be merged. Where, $Ø_{max}$ and $Ø_{min}$ 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.

Figure 1 shows an example of merging and splitting for K=3. If there is a group in which the total number of users, *u*, is greater than $Ø_{max}$, 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 *u* is less than $Ø_{min}$, 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 paper first-fit decreasing (FFD) bin-packing algorithm and the least loaded (LL) are used. Both these algorithms are given the same input and the results are compared for various *n* values.

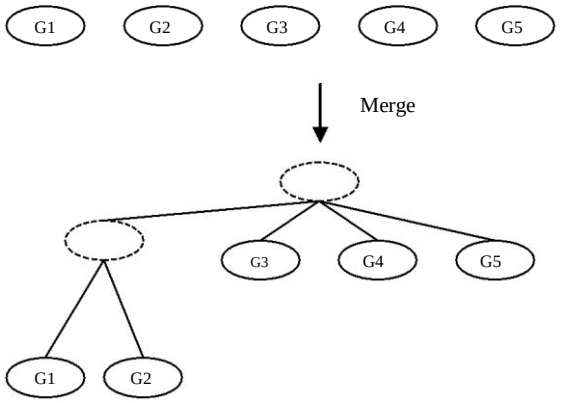

Figure 2: An example of server merge

4. ALGORITHMS

In this section, we present two algorithms that are evaluated in our experiments. 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.

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 Figure 3. 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 $Ø_{min}$, 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 servers. In order to keep the load as balanced as possible, the server with more number of users are added into higher level nearer to the root (i.e., level) while the server with less number of users are added into the lower level. Fig. 2 illustrates a case of merging five servers with a branching factor of 4. If G1 and G2 are two groups with lesser number of users, these two groups are added into the second level and the larger groups 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 Figure 3. FFD receives *n* existing servers and sorts them in descending order of utilizations of a certain resource.

TABLE 1
NOMENCLATURE USED IN THE PAPER

Symbol	Definition
ρ	*Total average number of concurrent users*
g	*Number of multicast groups*
$Ø_{max}$	*Maximum threshold for the number of users in a server*
$Ø_{min}$	*Minimum threshold for the number of users in a server*
δ	*Utilization*
μ	Service rate
n	Number of existing servers
m	Number of destination servers
u	Number of users in the server
K	Branching factor

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, ..., *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.

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.

```
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
```

Figure 3: FFD algorithm

4.2 Least Loaded algorithm

The LL algorithm works on the principle of load balancing. The LL algorithms attempts to balance the load between servers by assigning incoming jobs to the least – loaded server. In server packing, an existing server with a high utilization is packed into a destination server with a low utilization. Figure 4 shows the LL algorithm that addresses the server packing problem. The function LB ($\{s_1, ...s_n\}$) in the figure 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.

Figure 4: LL algorithm

```
sort existing servers to {s1,.....sn} in descending order;

m ← LB ({S1,....Sn});
while true do
for j ← 1 to m do
Xj ← {} /* initilization *?
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
```

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.

5. RESULTS AND DISCUSSIONS

In this section, we present the results from an extensive set of experiments to investigate the performance of the algorithms under study.

The algorithms are systematically evaluated across the wide spectrum of distribution parameter values for virtual server load and node capacity to give a clear view of the performance of the algorithms.

The performance metrics considered are:

- Consolidation Efficiencies
- Destination Servers
- Convergence Time
- Total Workload moved
- Success Ratio and
- Response Time

In a multiserver network of computing hosts, the performance of the system depends crucially on dividing up work effectively across multiple server nodes. The random arrival of users at each server is likely to bring about uneven server loads in such a system. Dynamic load balancing algorithms compared to bin packing algorithms have the potential to perform well under heavy loads. Naturally dynamic load balancing strategies are more complex and the overheads involved are much more. But one can not negate their benefits. Load balancing is found to reduce significantly the mean and standard deviation of job response times, especially under heavy and/or unbalanced workload. The performance is strongly dependent upon the load index. The reduction of the mean response time increases with the number of hosts, but levels off beyond a few tens of hosts.

n	Algorithm	*m*	*m/LB*	Convergence Time (sec)
50	FFD	39.6	1.34	0.061
	LL	37	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	1.09	0.188
200	FFD	188	1.14	0.127
	LL	171	1.09	0.284
250	FFD	217	1.08	0.142
	LL	203	1.05	0.323

Table 2: Comparison of average number m of destination servers offered by FFD and LL for various n values

The values m / LB 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). The results show that while m increases linearly with n, LL algorithm results in the better m values compared to FFD algorithm. Similarly, the convergence time for LL algorithm is better than FFD.

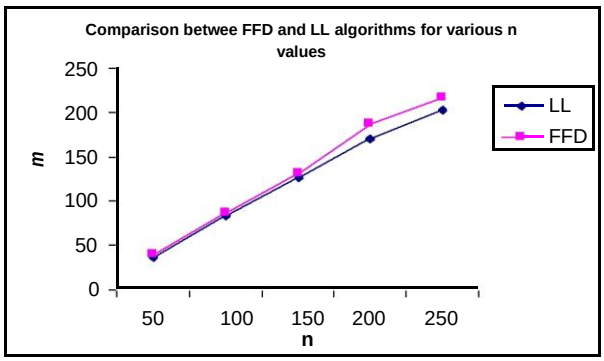

Figure 5: Comparison of FFD and LL based on m

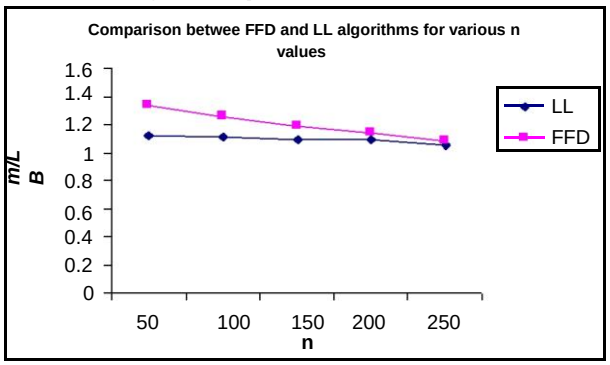

Figure 6: Comparison of FFD and LL based on m/LB

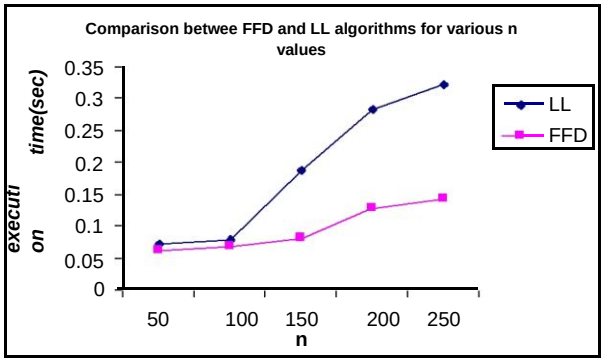

Figure 7: Comparison of FFD and LL based on convergence time

Response time is a function of the CPU requirements of the components comprising the application, the number of remote messages exchanged by these components, and the current load on required resources. Our objective function is to improve performance which is simply defined as minimization of the system average response time. For this, the number of application remote messages exchanged has to be kept as low as possible. For each remote message exchanged, we model the cost incurred by adding a delay to the time consumed by the message originator. Application performance (response time) is modeled as described in Fig.5.

In the figures, the workload moved performance result is plotted in the form of the total load moved as a fraction of the total workload of the system when the algorithms successfully terminate. The success ratio is defined to be the percentage of the problem instances for which feasible solutions are found among all problem instances.

Given a server S, comprising n users: $n_1 \cdots n_n$:
response time
(S(i))=wait_time(S(i))+cpu_required(S(i))+itc_cost(S(i))+system_cpu(S(i))
where: wait_time(S(i))=start_time(S(i))-arrival_time(S(i))
itc_cost(S(i))=number_of_key_exchanges(S(i))
[itc-information transfer cost]
system_cpu(S(i))=cpu time „stolen" for system activities during the components" execution.

Given a server with a heuristic algorithm (HA) and n users, performance (HA) = response_time

$$(HA) = 1/n \sum_{k=1}^{n} \text{response\_time(server } S_k)$$

Figure 8: Calculation of response time

n	Algorithm	Success Ratio (%)	Moving Workload (%)	Response Time(ms) - Split	Response Time(ms) - Merge
50	FFD	100	19	18.5	11.3
	LL	99.8	20.3	11.2	10.7
100	FFD	99.3	19.8	19.1	11.9
	LL	99.6	20.8	11.9	10.9
150	FFD	98.7	21	19.9	12.4
	LL	99.5	21.6	12.3	11.4
200	FFD	98.5	21.3	20.3	12.6
	LL	99.5	21.9	13.1	11.8
250	FFD	98.5	21.5	21.5	13
	LL	99.5	22.1	13.5	12.1

Table 3: Comparison of average number m of destination servers offered by FFD and LL for various n values

Load balancing is still very effective when a large portion of the workload is immobile. All servers, even those with light loads, benefit from load balancing. System instability is possible, but can be easily avoided. The Least-Loaded algorithm produced average response times representing 34.8% of the average response times produced by the FFD bin packing algorithm.

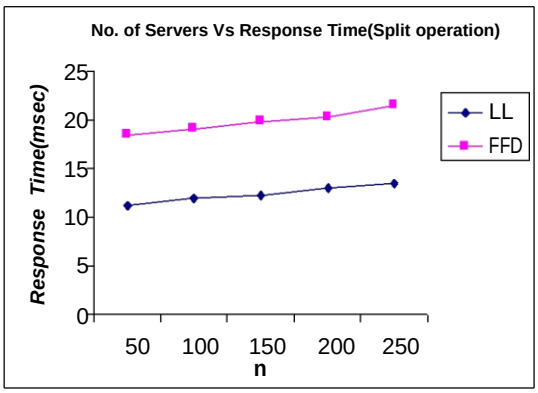

Figure 9: Comparison of FFD and LL based on response time (split operation)

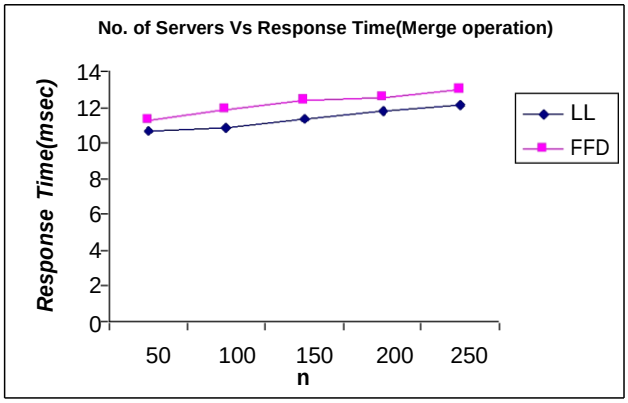

Figure 10: Comparison of FFD and LL based on response time (merge operation)

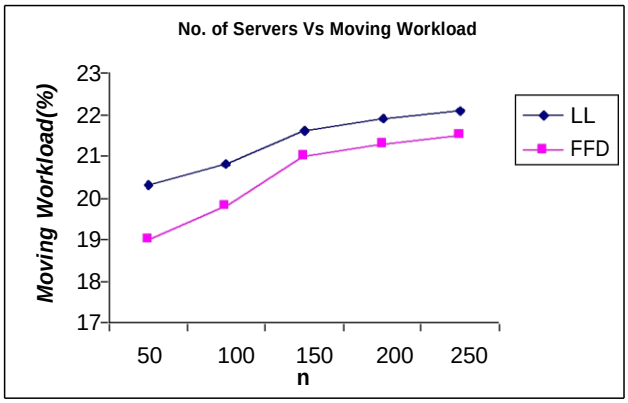

Figure 11: Comparison of FFD and LL based on success ratio

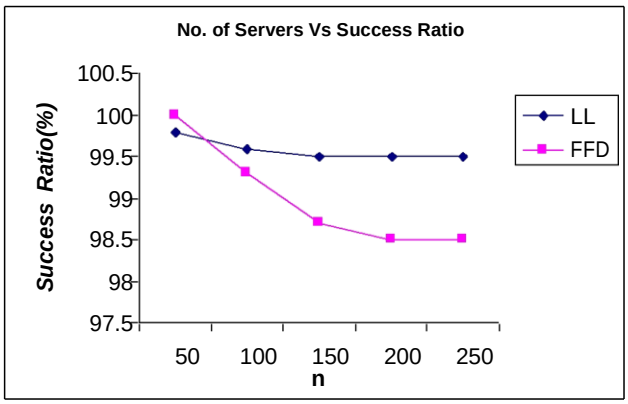

Figure 12: Comparison of FFD and LL based on moving work load

6. 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].Valeria Cardellini, Michele Colajanni, Philip S. Yu, "Redirection Algorithms for Load Sharing in Distributed Web-server Systems," icdcs, pp.0528, 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99), 1999

[2].Yao Zhao; Fangchun Yang , "A Dynamic Load Balancing Algorithm for Distributed SLEE in Mobile Service Provisioning", International Conference on Wireless Communications, Networking and Mobile Computing, 2006 WiCOM 2006, Volume 1, Issue 4, 22-24 Sept. 2006 Page(s):1 – 4

***[3].Yong Meng Teo, "**Comparison of Load Balancing Strategies on Cluster-based Web Servers, SIMULATION, Vol. 77, No. 5-6, Pages 185-195, 2001.*

[4].G. Shen, S. K. Bose, T. H. Cheng, C. Lu and T. Y. Chai, —Efficient heuristic algorithms for light-path routing and wavelength assignment in WDM networks under dynamically varying loads , Computer Communications, Volume 24, Issues 3-4, Pages 364-373, 2001.

[5] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional bin packing problem with fixed orientation operations research letters, Vol. 35, No.3, pp. 357-364, 2007

[6].A Caprara and P. Toth. Lower bounds and algorithms for two dimensional vector packing problem. Discrete Applied Mathematics, 111:231-262, 2001

[7].Lodi, S. Martello, and D. Vigo. Recent Advances on two dimensional vector packing problem Discrete Applied Mathematics, 123:379-396, 2002.

[8].Spellmann, K. Erickson, and J. Reynolds. Server consolidation using performance modeling. IT Professional, 5:31-36, 2003.

[9]. D L Eager , E D Lazowska , J Zahorjan, —A comparison of receiver-initiated and sender-initiated adaptive load sharing, Performance Evaluation, v.6 n.1, p.53-68, 1986.

[10]. Arthur P. Goldberg, Gerald J. Popek , Stephen S. Lavenberg, —A Validated Distributed System Performance Model, Proceedings of the 9th International Symposium on Computer Performance Modelling, Measurement and Evaluation, p.251-268, May 25-27, 1983

[11].Hać,"A Distributed Algorithm for Performance Improvement through Replication and Migration," Proc. IEEE Computer Networking Symposium November 17-18, 1986, Washington, D.C., pp. 163--168.

[12] Asser N. Tantawi, Don Towsley, Optimal static load balancing in distributed computer systems, Journal of the ACM (JACM), vol.32, no.2, pp .445-465, 1985.

[13].Y.-T. Wang, and R. J. T. Morris, "Load Sharing in Distributed Systems," IEEE Transactions on Computers, Vol. C-34, No.3, pp. 204—217, 1985.

[14].B. S. Baker, —A new proof for the first-fit decreasing bin-packing algorithm, J. Algorithms, vol. 6, pp. 49–70, 1985.