ISSN:0975-9646

V.Hema et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 7 (4) , 2016, 2059-2062

# Request Distribution Strategies in Cluster Based Network Servers

**[1]V.Hema [2]Dr. K.Kungumaraj**

*[1]Research Scholar, Mother Teresa Women's University, Kodaikanal.*
*[2]Assistant Professor, Department of Computer Science,*
*Arulmigu Palaniandavar Arts College For Women, Palani.*

**Abstract: As distributed applications and services are getting popular now a days, Web servers play a central role in the telecommunications infrastructure. Cluster-based Web servers are increasingly adopted to host a variety of network-based services. So performance improvement of these servers is necessary. The cluster based architecture consists of front-end dispatcher and several back-end servers. Front end is responsible for request distribution. The back-end nodes are responsible for request processing. They should provide reliability, availability and efficient services. So performance improvement of these servers is necessary. Load balancing is one of the best efficient methods for performance improvement of cluster system. It is often desirable to isolate the performance of different classes of requests from each other. Our objective is to deliver better services to high priority request classes without over-sacrificing low priority classes. The main objective is to minimize the response time of requests that need intra cluster communication.**

**Keywords : Load balancing, Web Servers, Clusters, Load Dispatcher**

## 1. INTRODUCTION

Communication networks today have become essential for big business. The network traffic increasing rapidly requires an increase in backbone networks capacity and needs to be upgraded frequently. The web server hosts the pages, scripts, programs and multimedia files and serves them using protocols. To increase web server scalability more servers needs to be added to distribute the load among these server cluster. A cluster consists of a number of nodes connected by a high speed LAN. There are three main components in the cluster (1) Dispatcher (2) Distributer and (3) Server. The communication between components on different cluster nodes takes place using persistent TCP control connections. These connections also serve to detect node failures. The load distribution among this cluster server is known as load balancing. As a single web server cannot handle the traffic, a load balancer is required to balance the traffic load across multiple servers.

Request distribution and load balance are essential techniques for web server clusters. Cluster based server has been proven to be an efficient and cost effective alternative to build a scalable, reliable and high-performance Internet server system. For cluster computing the network dispatching technology for client's requests is an important issue. The best way to address both the scalability and reliability problems in web clusters is to deploy a totally decentralized architecture.

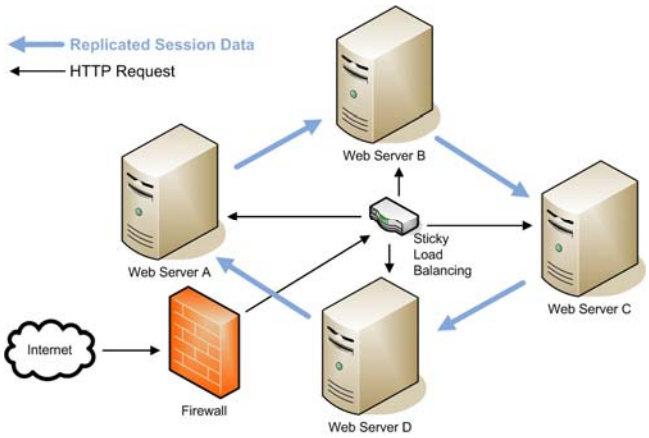


**Figure-1: Load balancing among multiple web servers**

A cluster-based server consists of a front-end dispatcher and multiple back-end servers. The dispatcher receives incoming jobs, and then decides how to assign them to back-end servers. The back end servers serve the jobs according to some policies. The strategies for load balancing on back end servers depend on the amount of work and the number of jobs assigned. The switch acts as the initial interface between the cluster nodes and the Internet, and distributes the incoming requests to the servers, trying to balance the load among them. The following figure shows the cluster theory.

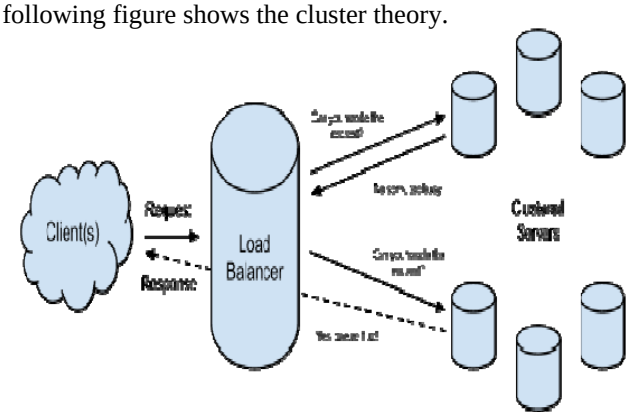


**Figure-2: Cluster Theory**

The clients issue their requests, the load balancer forwards these request to the clustered server which in turn replies with yes or no response. The load balancer processes this response and send back to clients. As the number of client requests increases effective strategies must be followed to balance the load among the servers. In this paper we present the various strategies for request distribution in cluster based network servers.

## 2. REQUEST DISTRIBUTION STRATEGIES

The following strategies provide various methods for effective load balancing among servers.

### 2.1 Content Aware Request Distribution

Cluster-based servers employ a specialized front-end node that acts as a single point of contact for clients and distributes requests to back-end nodes in the cluster. The front-end distributes requests such that the load among the back-end nodes remains balanced. With content aware request distribution, the front-end additionally takes into account the content of service requested when deciding which back-end node should handle a given request. Content aware request distribution can improve scalability, flexibility and can give significant performance improvements. It also affords simplicity and limit the scalability of the cluster.

Figure-3 depicts a simple client-transparent mechanism. An HTTP proxy running on the front-end accepts client connections with all back-end nodes. When a request arrives on a client connection, it is assigned to according to request distribution strategy and the request is forwarded to appropriate back-end connection. When the response arrives from backend node, the front end proxy forwards the data on the client connection. This approach is simple and no modification is required on cluster node. Because of over head incurred for forwarding all response data from back-end server to the clients. TCP splicing has low overhead but requires modifications to the OS Kernel of the front end node. TCP hand off mechanism was introduced to enable the forwarding of back-end responses directly to the clients without passing through the front-end intermediary. The TCP handoff is totally transparent from the client's point of view. Since it operates on transport-level streams, clients can never be aware of being redirected. TCP handoff provides higher scalability than TCP splicing as it eliminates the forwarding overhead of response data.

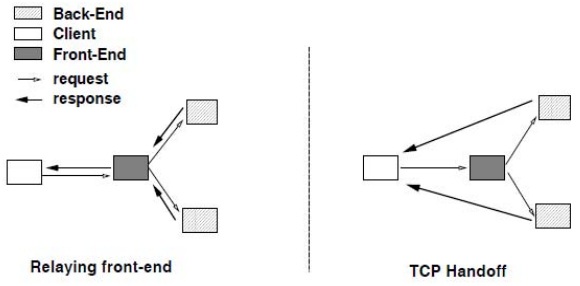


**Figure-3: Mechanisms for request distribution**

There are three main parts in content aware request distribution scheduling system. i. Design of dispatcher module ii. LARD scheduling policy iii. CASS. The following figure 4 illustrates the design of the network dispatcher. The pseudo-server module provides the function of listening on multiple known ports at the same time. The packet parser module achieves generality by providing a common interface for developers to add other packet parser for different network services.

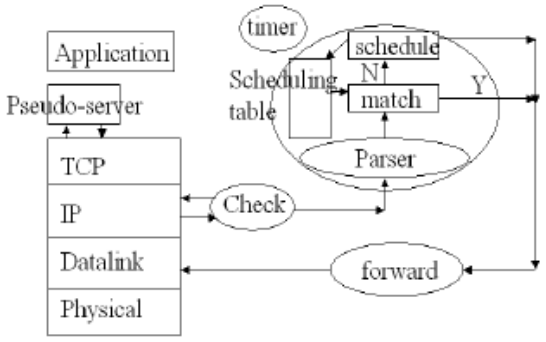


**Figure-4: Design of the Network Dispatcher**

In LARD scheduling policy, the allocation must be decided a prior, and the data must be allocated to different node servers. It can only support static web services. CASS supports many TCP-based network services. It can increase the node server's main memory cache hit rate and enhance the cluster's performance. The process delay of CASS is compared with common network delay.

### 2.2 Time-Window Based Request Distribution Strategy

In order to overcome the weakness of existing request distribution strategies for server cluster in average response time and computation cost, a request distribution strategy based on static time interval is used. Its basic idea is that it divides the update interval into several subintervals and introduces randomness into the selection of server node for the requests arrived in a subinterval. It can help to improve cache utility while keeping workload balance among the servers in the cluster. The throughput requirement of storage subsystem is relieved, and the system utility is improved. Round trip time can give better and relatively accurate delay experienced in path and to some extent; a lower RTT indicates higher available bandwidth. However, it is very dynamic in nature; it changes quickly over relatively short period of time. It has much more variation for different clusters compared to hop count; it gives better path information between client and cluster. On the downside, it is relatively costlier to measure and requires more frequent refreshes.

## 3. CO-SCHEDULING ARCHITECTURE

Co-scheduling is the principle for concurrent systems of scheduling related processes run on different processors at the same time (in parallel). There are three types of co-scheduling: explicit co-scheduling, local scheduling and implicit or dynamic co-scheduling. Explicit co-scheduling requires all processing to actually take place at the same time, and is typically implemented by global scheduling across all processors. A specific algorithm is known as gang scheduling. Local co-scheduling allows individual processors to schedule the processing independently. Dynamic (or implicit) co-scheduling is a form of co-scheduling where individual processors can still schedule processing independently, but they make scheduling decisions in cooperation with other processors. Figure-5 shows a co-scheduler model.

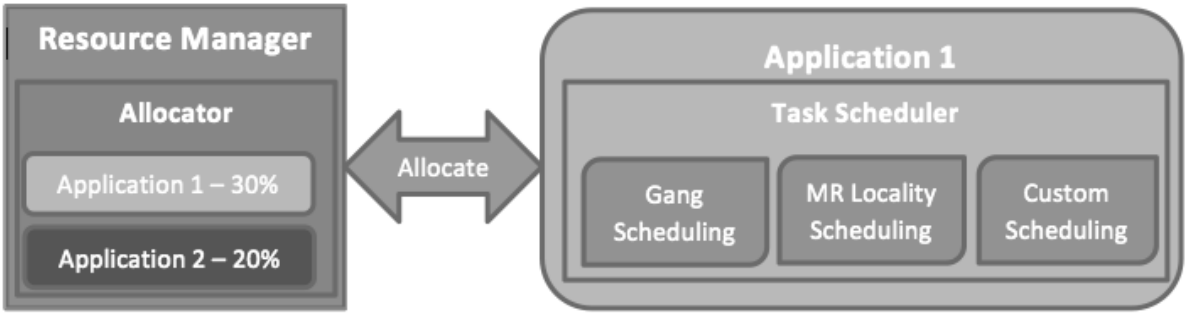


**Figure-5: Co-scheduler Model**

Unlike tightly coupled multiprocessors, scheduling processes of a parallel job onto various nodes of a cluster is more challenging due to the individual node autonomy. All co-scheduling algorithms rely primarily on one of two local events. (i) Arrival of a message (ii) Wait for a message to determine when and which process to schedule. Co-scheduling algorithms reduce the execution time of parallel applications. Each node in the cluster is aware of the cache and load information of other nodes. A remote request is forwarded; the main process puts a request in a queue. Both sender and receiver are scheduled simultaneously for efficient communication.

## 4. LARD STRATEGIES

The goal of LARD is to combine good balancing and high locality. Locality aware request distribution improve locality in the back-end's cache, a simple front-end strategy assigns request for all targets to a particular back-end. The cache in each back-end should achieve a much higher hit rate. LARD maintains mappings between targets and back-end nodes. To achieve a balance between load distribution and locality, LARD uses cost-balancing, cost-locality and cost-replacement.

The locality based distribution policy at the distributor and strives to increase the memory hits at the backend server's rather than the disk latency. The distributor maintains a table of the data types available at the backend servers' memory. The data types are assigned to the backend servers based on the initial server/data partitioning and are initially distributed evenly across the servers. When a new request arrives at the distributor, its data type is looked up in the distributor table and the corresponding server is identified. The request is forwarded always to that server for that particular data type.

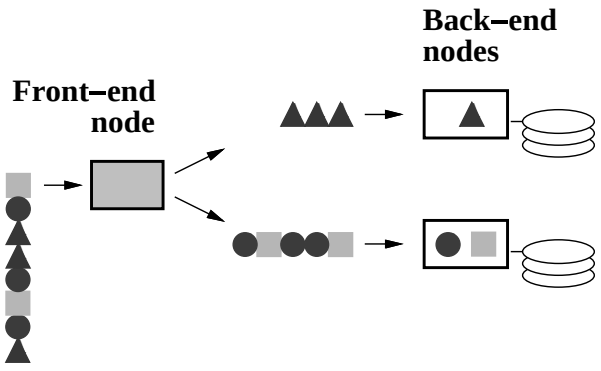


**Figure-6: LARD**

Figure-7 presents pseudo-code for the basic LARD. The front-end maintains a one-to-one mapping of targets to back-end nodes in the server array. When the first request arrives for a given target, it is assigned a back-end node by choosing a lightly loaded back-end. When a node is overloaded, the target is assigned a new back-end node from the current set of lightly loaded nodes. A node's load is measured as the number of active connections.

```
while (true)
fetch next request r;
if server[r.target]=null then
s, server[r.target] ← {least loaded node};
else
s ← server[r.target]
if (s.load > T_high && Exist node with load < T_low) || s.load
>= 2*T_high then
s, server[r.target] ← {least loaded node};
send r to s;
```

**Figure-7 : The Basic LARD Strategy**

$T_{low}$: the load below which a back-end is likely to have idle resources.

$T_{high}$: the load above which a node is likely to cause substantial delay in serving requests.

## 5. PERFORMANCE ANALYSIS

The intuition for the basic LARD strategy is as follows. The distribution of targets when they are first requested leads to a partitioning of the namespace of the database, purely aiming at locality. It also derives similar locality gains. We re-assign targets when there is a load imbalance. The front-end limits the total connections handed to all back-end nodes to the value $S = (n-1)*T_{high}+T_{low}-1$, where n is the number of back-end nodes. Setting S to this value ensures that at most n-2 nodes can have a load >= $T_{high}$ while no node has load < $T_{low}$.

The load difference between old and new targets is at least $T_{high}-T_{low}$. The max load imbalance that can arise is $2T_{high}-T_{low}$ . The setting for $T_{low}$ depends on the speed of the back-end nodes. Choosing $T_{high}$ involves a tradeoff. $T_{high}-T_{low}$ should be low enough to limit the delay variance among the back-ends to acceptable levels, but high enough to tolerate limited load imbalance without destroying locality. A single target causes a back-end to go into an overload situation. We should assign several back-end nodes to serve that document, and to distribute requests for that target among the serving nodes.

## 6. CONCLUSION

LARD strategy can achieve high cache hit rates and good load balancing in a cluster server. It also provides higher through put, better CPU utilization and reduced disk access. It is scalable at low cost. But LARD strives to improve cluster performance by simultaneously achieving load balancing and high cache hit rates at the back-ends. With LARD, the effective cache size approaches the sum of the individual node cache sizes. Thus, adding nodes to a cluster can accommodate both increased traffic due to additional CPU power and larger working sets due to the increased effective cache size.

## REFERENCES

[1]. Vivek S. Pai, Mohit Aron, Gaurov Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, Erich Nahum, “Locality-aware request distribution in clusterbasednetwork servers,” Proceedings of the eighth international conference on Architectural support for programming languages and operating systems, p.205-216, October 02-07, 1998, San Jose, California, United States

[2] E. V. Carrera, S. Rao, L. Iftode, and R. Bianchini. "User-Level Communication in Cluster-Based Servers". Proceedings of the 8th IEEE International Symposium on High-Performance Computer Architecture (HPCA 8), February 2002.

[3] M. Aron, D. Sanders, P. Druschel, etc, Scalable,Content-Aware Request Distibution in Cluster-Based Network Servers., Proceedings of 2000 USENIX Annual Technical Conference, 2000.

[4] T. 13risco. DNS Support for Load Balancing. RFC 1794, Apr. 1995.

[5] E. Pinheiro, R. Bianchini, E. V. Carrera and T. Heath, “Dynamic Cluster Reconfiguration for Power and Performance.” Kluwer Academic Publishers,2002.

[6] K. Rajamani and C. Lefurgy, “On evaluating request-distribution schemes for saving energy in server clusters,” in Proc. Intl. Sym. Performance Analysis of Systems and Software, March 2003.

[7] Teo Y.M. and R. Ayani. \Comparison of Load Balancing Strategies on Cluster-based Web Servers", Simulation, 77(5-6), 185-195, November-December 2001.

[8] Riska, A., W. Sun, E. Smirni and G. Ciardo. \AdaptLoad: E_ective Balancing in Clustered Web Servers Under Transient Load Conditions," 22nd International Conferenceon Distributed Computing Systems (ICDCS'02), 2002.

[9] Krishnamurthy, B. and J. Rexford. Web Protocols and Practice : HTTP 1.1, Networking Protocols, Caching, and Tra_c Measurement, Addison-Wesley, 2001.

[10] Ciardo, G., A. Riska and E. Smirni. \EquiLoad: A Load Balancing Policy for Clustered Web Servers". Performance Evaluation, 46(2-3):101-124, 2001.