DISPATCHING REQUESTS IN PARTIALLY REPLICATED WEB CLUSTERS
An Adaptation of the LARD Algorithm

Jose Daniel Garcia, Laura Prada
Computer Architecture Group, University Carlos III of Madrid, Avda Universidad Carlos III, 22, Colmenarejo, Madrid, Spain

Jesus Carretero, Felix Garcia, Javier Fernandez, Luis Miguel Sanchez
Computer Architecture Group, University Carlos III of Madrid, Avda de la Universidad, 30, Leganes, Madrid, Spain

Keywords: Web cluster, content replication, reliability.

Abstract: Traditional alternatives for Web content allocation have been full replication and full distribution. An hybrid alternative is partial replication where each content element is replicated to a subset of server nodes. Partial replication gives advantages in terms of balancing reliability and storage capacity. However, partial replication has architectural implications. In this paper we present a Web cluster architecture which may be used in single switched Web clusters and multiple switched Web clusters. We present an algorithm for Web content allocation which determines the number of replicas for each content based on its relative importance and that performs the allocation keeping in mind resource constraints in clusters with heterogeneous storage capacity. We also provide an adaptation of the LARD algorithm for request dispatching that copes with the fact that contents are partially replicated. Our evaluations show that performance of partial replication solutions is comparable to performance of traditional fully replicated solutions.

1 INTRODUCTION

During the last years the demand of high performance Internet Web servers has increased dramatically. The target is a solution which is scalable in both performance and storage capacity, while reliability is not compromised. The traditional standalone Web server approach presents severe scalability limits which may be partly mitigated by means of hardware scale-up (Devlin et al., 1999) which may be viewed as a short term solution. Web server performance may also be improved by acting on the operating system (Banga et al., 1998; Pai et al., 2000) or on the Web server software itself (Pai et al., 1999; Shukla et al., 2004).

Another approach is the distributed Web server (Cardellini et al., 2002) where a set of server nodes are used to host a Web site. In those systems, performance scalability may be achieved by adding new server nodes. Different architectures have been proposed for distributed Web servers. A common feature to those architectures is the fact that the set of server nodes offer a single system image to clients. Existing solutions may be classified in three families:

Cluster based Web Systems. In this solution (also known as *Web Cluster*) server nodes own IP addresses are not visible to clients. Instead, clients use a virtual IP address which corresponds to the address of some request distribution device or *Web Switch*. The Web switch (Aron et al., 1999; Schroeder et al., 2000) receives requests from clients and sends each request to a server node.

Virtual Web clusters. All the server nodes share a single common public IP address (Vaidya and Christensen, 2001). Each node receives all messages, but every request is discarded by all nodes except one.

Distributed Web systems. Each node has its own different publicly visible IP address (Aversa and Bestavros, 2000; Cardellini, 2003). Request distribution is made by a combination of dynamic DNS (Brisco, 1995) and request redirection.

Usually, content allocation has been based on full content replication (where every file is replicated into every server node) or on full distribution (where every file is stored in one and only one server node).

With full replication (Kwan et al., 1995; Devlin et al., 1999) the system is highly reliable and request distribution is easy to implement, as each request may be served by any server node. On the other hand, stor-

Daniel Garcia J., Prada L., Carretero J., Garcia F., Fernandez J. and Miguel Sanchez L. (2007).
DISPATCHING REQUESTS IN PARTIALLY REPLICATED WEB CLUSTERS - An Adaptation of the LARD Algorithm.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 141-149
DOI: 10.5220/0001265201410149

age scalability is minimal, as the full storage capacity is limited by the server node with the lowest capacity. Furthermore, adding new server nodes to the system does not increase storage capacity.

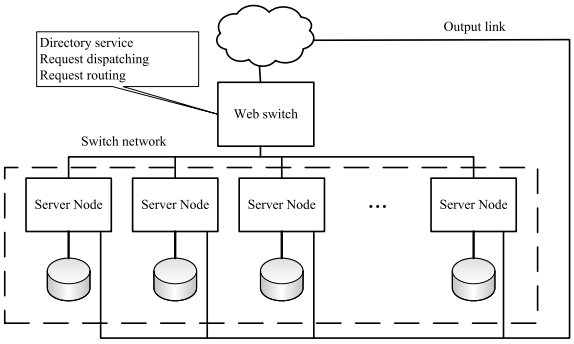

Figure 1: Architecture of a SSWC.

With full distribution the system gives a lower reliability (a node failure makes some content unavailable) and request distribution needs to use some sort of directory service (Baker and Moon, 1999; Apostolopoulos et al., 2000) to determine which node is storing the requested file. On the other hand, storage scalability is maximum, as adding a new node means increasing the total amount of available storage.

A third alternative is partial replication (Li and Moon, 2001; Garcia et al., 2003; Zhuo et al., 2003; Tse, 2005), where each file is replicated in a possibly different subset of the server nodes. With partial replication reliability is higher than with full distribution and lower than with full replication. In terms of storage capacity, partially replicated solutions provide less capacity than fully distributed solutions, but far more than fully replicated solutions. However, an important difference with full replication is that partial replication provides storage capacity scalability (i.e. adding new server nodes increases the system storage capacity), while full replication does not.

In this paper we present a solution for partial replication of contents in Web clusters. The paper is organized as follows. Section 2 discusses alternatives for the general architecture of a distributed Web server with partial replication and justifies the selection of the Web cluster architecture, giving design details for single switch and multiple switch variants. Section 3 presents our algorithm to determine the number of replicas for each element of a Web site and to allocate them to a set of server nodes. Section 4 presents an adaptation of the well known LARD algorithm for the case of partial replication of contents. Section 5 shows our evaluation results. In section 6 we summarize our conclusions. Finally section 7 outlines our future work.

2 ARCHITECTURAL ALTERNATIVES

Partial replication restricts the architectural alternatives for a distributed Web server. When a Web request arrives into the system, that request cannot be sent to any node in the system, as not every node stores every content. Besides, every file is not stored in a single node. Instead, it is replicated in a set of server nodes. Thus, a mechanism to determine that set of server nodes is needed.

Virtual Web Clusters cannot be easily integrated with the idea of partial replication (Garcia et al., 2006b). In that case, every request reaches to every node of the cluster. Nodes not storing the requested content may ignore the request. But if two or more nodes store the requested content it is not simple to determine which node should take the responsibility of answering the request, because most implementations make their selection using a hash function based only on client IP address and port. Such hash functions may lead to select a node not containing the requested file. Furthermore, in most cases communication among cluster nodes is not feasible, and that fact excludes the possibility that a node may notify others when it takes the responsibility of taking in charge of a request and its response.

In a *Distributed Web System* any request may reach any node, but it is guaranteed that one request reaches to one and only one node. However, there is no guarantee that the node receiving the request contains a replica of the requested node. In that case the only solution is that the node receiving the request performs a redirection to another node, which effectively contains the requested resource. This fact introduces additional complexities in the load balancing mechanisms.

Although it is not impossible to integrate the partial replication strategy into a *Virtual Web Cluster* or into a *Distributed Web System*, we consider that such strategy can be more easily integrated into a *Web Cluster*. The main difference of a *Web Cluster* with other architectures is that it has a single point where every request arrives (the *Web Swtich*). In case of partial replication, the Web Switch must be a content aware one (Cardellini et al., 2002). That is, the switch must operate at level 7 of the protocol stack and it must parse each request before deciding to which server node that request will be routed. In a previous work (Garcia et al., 2006a) we showed that a Web Cluster with partial replication using a small amount of replicas per file offers a reliability equivalent to that of a fully replicated system and, at the same time, it offers a much higher storage capacity.

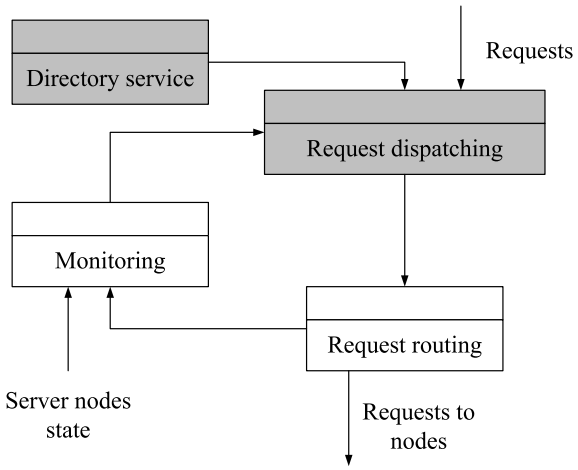

Figure 2: Internal architecture of Web Switch software for a SSWC.

However, reliability is negatively affected by the fact of using a single *Web Swtich*, which is a single point of failure. That flaw is mitigated by using more than one Web Switch (2 or three are enough).

2.1 Single Switched Web Cluster Architecture

In a Single Switched Web Cluster (SSWC), every request arrives into the Web Switch. The Web Switch is responsible for the selection of the node which must serve the request (through a dispatching algorithm). The Web Switch is also responsible for sending the request to the selected node (through a request routing algorithm). Existing solutions do not apply, because they usually assume either that every file is replicated in every node (full replication of contents) or that every file is stored in one and only one node (full distribution of contents). However, the general architecture is very similar to that of a standard Web Cluster (see Fig. 1) where every server node has two network interfaces: one connected to the Web Switch network, and another one connected to a high bandwidth output link.

In case of partial replication the dispatching algorithm is dependent of the exact set of nodes effectively containing the requested file. Thus, the *Web Switch* must have the knowledge of the mapping of each file to nodes (the set of nodes where it resides). It is for that reason that content blind routing is not applicable. As the dispatching algorithm must use the allocation information, some sort of *directory service* is needed to keep such information. Although a generic directory service could lead to excessive delays, it is possible to build low demanding resources data structures with lower delays (Apostolopoulos et al., 2000). As an example, Luo et al. (Luo et al., 2002) implemented a data structure using URL formalization and multilevel hash tables. Their solution stored the information related to 76000 files in 540 KB and the time needed for a query was about 1.12 microseconds.

Once the dispatching algorithm has selected a server node for a request, it is necessary to use some request routing mechanism to effectively send the request to the selected node. As we have already stated, request routing needs to be content aware. The routing mechanism may be implemented at the application level (e.g. TCP gateway (Casalicchio and Colajanni, 2001)) or at the operating system kernel level (e.g. TCP splicing (Maltz and Bhagwat, 1999) or TCP handoff (Pai et al., 1998)).

Several modules are needed in the Web Switch software, as depicted in Fig. 2:

- **Request dispatching** Receives requests and selects the node which must serve each request. The request dispatching algorithm uses replica allocation information (given by *directory service*) and node state information (given by *monitoring* module).
- **Request Routing** Receives dispatched requests and performs request routing to selected server node. We propose the usage of TCP handoff for efficient request routing.
- **Directory service** Efficiently keeps replica allocation to node information.
- **Monitoring** Keeps node state information (provided periodically by each node) which is used by the *request dispatching* module. It also keeps trace of routed requests information (provided by the *request routing* module).

Modules in gray color are new (or need modification) in our architecture. A *directory service* module has to be added to keep track of the mapping of files to nodes. Besides the *request dispatching* module needs to be modified to use the information of the directory service.

2.2 Multiple Switched Web Cluster Architecture

SSWC architecture presents a reliability drawback. As they have a single switch, that element becomes in a single point of failure. That is, if there is a failure in the Web switch, the full system fails. To avoid this flaw, we propose a modified architecture: the multiple switched Web Cluster (MSWC) based on the concept of *distributed Web switch*.

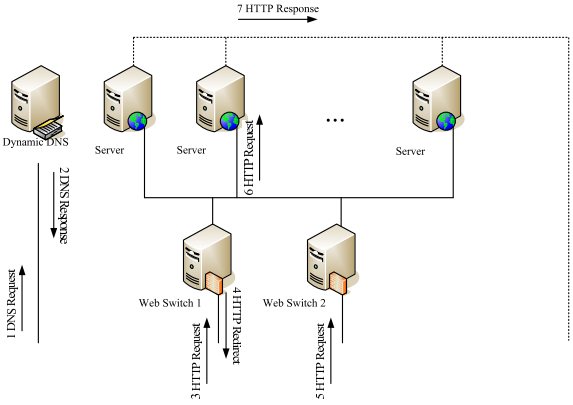

Figure 3: Request routing in a MSWC.

The main idea is the usage of a set of Web switches as front end of the Web cluster. This allows a high increase of the global reliability (Garcia et al., 2006a). To distribute request arrival among the Web switches, we use the combination of two mechanisms: dynamic DNS and request redirection among switches.

Dynamic DNS (Andresen et al., 1997) has been widely used in conjunction with *Distributed Web Systems* to share requests from clients among a set of publicly available nodes. However, this technique is of limited usefulness due to DNS caching, as several experiments (Colajanni et al., 1998) have shown.

To complement dynamic DNS, each Web switch may redirect some requests to another Web switch of the front end. In our system, when a Web switch is highly loaded, it redirects incoming requests to another Web switch. That redirection may be performed by means of standard HTTP redirection.

Fig. 3 shows the flow of a request in a MSWC. When a client starts a request, it first issues a DNS resolution request (1), which is answered by the DNS server (2) with the IP address of a Web switch in the front end of the cluster. The client then sends a HTTP request to the selected Web switch (3), which may answer with a redirection to another Web switch (4). In that case, the client resends the HTTP request to the newly selected Web switch (5). The Web switch selects a server node through its dispatching algorithm and resends the HTTP request to it (6). At last, the selected node answers the client (7).

For the solution to work properly, each Web switch must monitor its own state and, periodically, exchange that information with the other Web switches. This allows that each switch may have enough information to decide when it is able to transfer some requests to another switch to achieve some degree of load balancing at the front end level. It is important to remark that transferring a request to another switch has a lower cost than keeping it in the original switch. That is because transfer operation may be performed in a content blind manner at the TCP level of the protocol stack and may be easily integrated in the operating system kernel. When a switch takes the responsibility of managing a request it selects a server node and it routes the request to that server node.

The structure of the software running in each server node (see Fig. 4) is more complex than the one used for a SSWC. Main modules are listed below:

- **Switch routing filter** Decides if an incoming request may be processed by the current switch or if there is a better switch to serve the request. The decision is based on the load state of every switch. That information is provided by the state synchronization module.
- **Request dispatching** Receives requests not discarded by the switch routing filter and selects the node which must serve each request. The request dispatching algorithm uses replica allocation information (given by directory service) and system state information (given by state synchronization).
- **Request routing** Receives dispatched requests and performs request routing to selected server node. We propose the usage of TCP handoff for efficient request routing.
- **Directory service** Efficiently keeps replica allocation to node information. To reduce communication overheads we place a directory service replicated in every Web Switch of the system.
- **Monitoring** Keeps node state information (provided periodically by each node). It also keeps trace of routed requests information (provided by the request routing module). The information it gathers is provided to the state synchronization module to build an integrated system state view.
- **State synchronization** Periodically notifies the rest of Web switches about its own state. This allows that every switch has knowledge about the state of other switches. That information is integrated with information about server nodes provided by the monitoring module to help switch routing filter and request dispatching modules to make their own decissions.

3 REPLICA ALLOCATION

A key point to use a partial replication strategy is replica allocation. That means answering two questions:

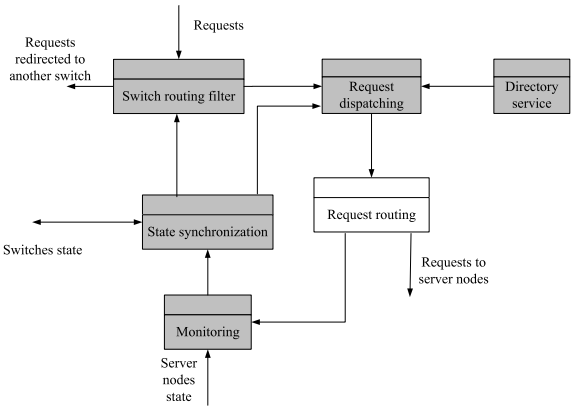

Figure 4: Internal architecture of Web Switch software for a MSWC.

1. How many replicas must be used for each element?

2. How must the replicas for different elements be distributed over the cluster?

For question 1 the most easy solution would be to use a fixed number of replicas for each element (e.g. three replicas per file). This solution does not takes into account the fact that some files do not need to be replicated as they are rarely accessed. Besides, having the same number of replicas for each file leads to problems when nodes have heterogeneous storage capacities. Having an architecture that allows heterogeneous storage capacities is an useful feature when scaling up nodes as it is not needed that every node has the same storage capacity.

To select the number of replicas to be used for each element, some criteria (or combination of criteria) is needed. That allocation criteria may be expressed as the relative importance of the element or file. One example for such criteria is access frequency to elements. In that case, more replicas could be used for highly accessed elements, using few replicas for elements seldomly accessed. Another example of criteria is relative importance for the content provider. In that case, important elements have more replicas to provide a better fault tolerance.

Let s_i be the size of element e_i, c_j the capacity of j-th server node and w_i the normalized weight for element e_i. A first approach is to assign for each element an amount of space proportional to its weight, and determine the number or replicas r_i that can fit in that space, as shown in (1). To ensure that at least one replica is assigned to every element, only the remaining space after assigning first replica is distributed among elements.

Table 1: Meaning of symbols used to express replica allocation algorithm.

Symbol	Meaning
N	Number of elements in Web site
M	Number of server nodes
w_i	Weight for i-th element
c_j	Storage capacity for j-th server node
s_i	Size of i-th element
r_i	Unadjusted number of replicas for i-th element
r_i^*	Adjusted number of replicas for i-th element

$$r_i = 1 + \frac{w_i \left(\sum_{j=1}^{M} c_j + \sum_{k=1}^{N} s_k \right)}{s_i} \quad (1)$$

However, (1) may lead in results of more than M replicas for an specific element. To correct this, we define r_i^* to adjust the number of replicas to a value in the integer interval $[1, M]$, as shown in (2).

$$r_i^* = \begin{cases} M & \text{if } r_i > M \\ \lfloor r_i \rfloor & \text{if } r_i \leq M \end{cases} \quad (2)$$

Once the number of replicas has been determined for each element, and in order to answer question 2, an algorithm is needed to allocate replicas to server nodes. For this, we use a greedy algorithm which allocates first larger files and afterward smaller ones. Algorithm 1 shows our approach which determines the number of replicas for each element and allocates them to server nodes, and Table 1 shows notation for the algorithm.

Algorithm 1 Greedy algorithm for replica allocation.

for $i = 1$ to N **do**
  $r_i \leftarrow 1 + w_i(\sum_{j=1}^{M} c_j + \sum_{k=1}^{N} s_k)/s_i$
  $r_i^* \leftarrow \min\{M, r_i\}$
  $j \leftarrow k \in [1, M] \,|\forall l c_k > c_l$
  for $l = 1$ to r_i^* **do**
    $a_{il} \leftarrow 1$
    $c_l \leftarrow c_l - s_i$
  end for
end for

4 REQUEST DISPATCHING ALGORITHM

In both architectures SSWC and MSWC request dispatching is a key element, as it receives a request and it selects the server node responsible for processing that request. We have adapted LARD (locality aware request distribution) (Aron et al., 2000) for the context of partial replication of Web contents. We name our version PLARD (Partial LARD). LARD tries to maximize the hits on file systems caches and Web server caches. To achieve this goal the algorithm tries to assign all the requests of a file to the same server node, when that node is below a certain load level. Only when the server node is highly loaded a new node is selected to serve requests to that file.

LARD algorithm assumes that every file may be found on every server node. That is, the algorithm is suitable for fully replicated Web clusters. In a partially replicated Web clusters some adaptations are needed:

1. If the number of replicas for a specific file is selected using the probability for that file to be requested, only one replica is used for such file. It is important to remark that it should be expected for a large number of files to be in that category, as many files are rarely accessed. For all those files the dispatching algorithm is not needed as there is only one server node capable of serving requests.
2. The algorithm must consider, for each file, only the server nodes where it is effectively stored. This means, that the set of nodes assigned to service a file must be a subset of the set of nodes effectively storing that file.

In Table 2 we show the notation we use for our PLARD algorithm.

Initially (see algorithm 2), a family of sets S is constructed where each set S_i contains the server nodes s_j where the file e_i is stored. Besides, a second family of sets U is constructed where every set U_i is initially empty. Every set U_i is used to store the server nodes assigned to service requests for an specific element e_i. Additionally, the value t_i^{mod} is set to 0 for each file e_i. That value is used to keep track of the latest instant in which the corresponding file has been modified.

When a request reaches the Web switch (see algorithm 3), the number of server nodes capable of servicing the request is determined ($|S_i|$). If the requested element may be only serviced by on node, the request is assigned to that node. Otherwise the corresponding U_i set is analyzed (i.e. the set of nodes currently assigned to service requests for that element).

Table 2: Meaning of symbols used to express PLARD algorithm.

Symbol	Meaning
N	Number of elements in Web site
M	Number no server nodes
a_{ij}	Element allocation matrix element (1 if element i is allocated to server j)
S_i	Set containing server nodes where the i-th element is stored
s_k^i	k-th element of set S_i
U_i	Set containing nodes currently servicing requests for element i
u_k^i	k-th element of set U_i
$load(s_k^i)$	Load level of k-th node in set S_i
t_i^{mod}	Instant when set T_i was modified last time
t	Current time.

Algorithm 2 Value initialization for PLARD.

```
for i = 1 to N do
  t_i^mod ← 0
  S_i ← ∅
  U_i ← ∅
  for j = 1 to M do
    if a_ij = 1 then
      S_i ← S_i ∪ {s_j}
    end if
  end for
end for
```

When set U_i is empty, the least loaded node of corresponding set S_i is selected and added to set U_i. When set U_i is non-empty, the least loaded node (s_sel) of set U_i is selected, if it is below a load threshold. If there is no node in set U_i below load threshold, a new node is selected to be added to set U_i. A special case is when all nodes containing replicas of the requested element are in set U_i (i.e. $|U_i| = |S_i|$). In such a case, the least loaded node in U_i is used to serve the request.

When load associated to an element decreases, an element is eliminated from the corresponding U_i set. To achieve this, whenever set U_i is updated the corresponding modification time t_i^{mod} is updated to current time. If after a timeout $t_r ev$ no change has happened to set T_i, the least loaded node of T_i is eliminated.

5 PERFORMANCE EVALUATION

Even if reliability and storage capacity are key issues for a Web server, these goals must be achieved without loss of performance or with a minimum loss. To

Algorithm 3 Request dispatching in PLARD.

if $|S_i| = 1$ **then**
  Select node s_1^i
else
  if $U_i = \emptyset$ **then**
    $s_{sel} \leftarrow s_j^i \in S_i | load(s_j^i) = \min_{s_k^i \in S_i} \{load(s_k^i)\}$
    $U_i \leftarrow U_i \cup \{s_{sel}\}$
    $t_i^{mod} \leftarrow t$
    Select node s_{sel}
  else
    $s_{sel} \leftarrow u_j^i \in U_i | load(u_j^i) = \min_{u_k^i \in U_i} \{load(u_k^i)\}$
    if $|U_i| = |S_i|$ **then**
      Select node s_{sel}
    else if $load(s_{sel}) > L_{MAX}$ and $\exists s_k^i | load(s_k^i) < L_{MIN}$ or $load(s_{sel}) \geq 2L_{MAX}$ **then**
      $s'_{sel} \leftarrow s_j^i \in S_i | load(s_j^i) = \min_{s_k^i \in S_i} \{load(s_k^i)\}$
      $U_i \leftarrow U_i \cup \{s'_{sel}\}$
      $t_i^{mod} \leftarrow t$
      Select node s'_{sel}
    else
      Select node s_{sel}
    end if
    if $|S_i| > 1$ and $t - t_i^{mod} > t^{rev}$ **then**
      $s_{elim} \leftarrow u_j^i \in U_i | load(u_j^i) = \max_{u_k^i \in U_i} \{load(u_k^i)\}$
      $U_i \leftarrow U_i - \{s_{elim}\}$
      $t_i^{mod} \leftarrow t$
    end if
  end if
end if

evaluate performance of our solutions we have built a simulation model using the OMNeT++ 3.0 framework (`http://www.omnetpp.org`).

For our simulations we have used a Web cluster with 16 server nodes. We have set up 800 client machines which are continuously performing Web requests to the cluster. Requests are routed using an one-way strategy (responses from server nodes use a different connection so they do not return through the Web switch). Table 3 shows the main simulation parameters used in the evaluations which are consistent with other works (Barford and Crovella, 1998; Casalicchio and Colajanni, 2001; Cardellini, 2003).

In our evaluation we have compared full replication of contents (FREP), where every file is replicated into every node, to partial replication (PREP) where files are partially replicated. In the latter case we have used file popularity as selection criteria. We use Zipf function as as popularity model (Breslau et al., 1999).

We have used two test scenarios: a SSWC and a MSWC with three Web switches. For each scenario we performed 35 simulation realizations with different random seeds. Table 4 shows mean service time and variance for SSWC and MSWC.

The obtained results show slight overheads in case of partial replication over the case of full replication. To estimate if there is difference in obtained results we have conducted an analysis of variance (ANOVA) with $\alpha = 0.05$ over our simulation results. In both cases, SSWC and MSWC, we obtained a probability that there is no difference in results of more than 99%.

Thus, there is no evidence, in our simulation experiments, that the usage of a partial replication strategy affects negatively performance. This is true for both, our experiments of a SSWC and for our experiment using a front-end of three Web switches. It is remarkable to note that, even if we admitted that performance is not equal differences in performance are below 0.5% which makes our solution very useful as it offers a very good tradeoff between reliability and storage capacity.

6 CONCLUSION

In this paper we have presented a Web cluster architecture which may be used in presence of partial replication. Our previous works have shown that partial replication allows to balance global reliability and storage capacity. In this work we have focused on the needed architecture for a cluster based Web system to provide services when contents are partially replicated. Using the Web cluster as base architecture allows that key changes are needed only in the Web switch component.

We have presented two variants of our architecture: one for single switched web clusters and another one for multiple switched web clusters. Both architectures share as a key element an efficient directory service to keep track of file to node mappings.

When using partial replication, two issues have to be solved: (1) the replica allocation strategy and (2) the request dispatching algorithm used to select the server node responsible to service every request. For replica allocation strategy we have presented an algorithm that keeps in mind storage resource constraints and gives more replicas to more relevant files. Relevancy of files may be defined in terms of popularity, relative importance of the content or any other organization defined criteria. It is remarkable to note that our algorithm may be used when nodes have heterogeneous storage capacities. For request distribution

Table 3: Simulation parameters used in the evaluation.

Parameter	**Distribution**
Number of embedded files	Pareto ($\alpha = 2.43, k = 1$)
Main files size (body)	Lognormal ($\mu = 7.63, \sigma = 1.001$)
Main files size (tail)	Pareto ($\alpha = 1, k = 10240$)
Embedded files size	Lognormal ($\mu = 8.215, \sigma = 1.46$)
Inter session time	Pareto ($\alpha = 1.4, k = 20$)
Requests per session	Inverse gaussian ($\mu = 2.86, \lambda = 9.46$)
Inactivity time	Pareto ($\alpha = 1.4, k = 1$)
Parsing time	Weibull ($\alpha = 1.46, \beta = 0.382$)

Table 4: Mean service time and correspondig variance for request processing in SSWC and MSWC.

Replication	**SSWC**		**MSWC**	
	Mean	Percentile-90	Mean	Percentile-90
FREP	5.423566866 s.	12.91512182 s.	5.437371853 s.	13.04278589 s.
PREP	5.423986327 s.	12.97181347 s.	5.437461929 s.	13.03745480 s.

we have adapted the LARD algorithm for the case of partial replication leading to our PLARD.

The performance evaluation suggest, that there is no significant difference in performance between full replication and partial replication. Moreover, even if we assume that there is a difference this would be below 0.5%. The main advantage of our solution is that is capable to offer a good balance among storage capacity and reliability.

7 FUTURE WORK

The solution we have presented if of static nature in the sense that replica allocation is performed at start phase and it is then frozen. However conditions of a Web site may vary when new contents are added, or the usage pattern is modified. It is for this reason that we plan to redefine our architecture so that replica allocation may be automatically updated incorporating properties of self-managed systems. This will be specially useful in cases where file popularity is used as replica allocation criteria.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Ministry of Science and Education under the TIN2004-02156 contract and Madrid Regional Government under contract UC3M-INF-05-003.

REFERENCES

Andresen, D., Yang, T., and Ibarra, O. H. (1997). Toward a scalable distributed www server on workstation clusters. *Journal on Parallel and Distributed Computing*, 42(1):91–100.

Apostolopoulos, G., Aubespin, D., Peris, V., Pradham, P., and Saha, D. (2000). Design, implementation and performance of a content-based switch. In *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, volume 3, pages 1117–1126.

Aron, M., Druschel, P., and Zwaenepoel, W. (1999). Efficient support for P-HTTP in cluster-based web servers. In *Proceedings of the 1999 USENIX Annual Technical Conference*, pages 185–198.

Aron, M., Sanders, D., Druschel, P., and Zwaenepoel, W. (2000). Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pages 232–336.

Aversa, L. and Bestavros, A. (2000). Load balancing a cluster of web servers: using distributed packet rewriting. In *Conference Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference (IPCCC 2000)*, pages 24–29.

Baker, S. M. and Moon, B. (1999). Distributed cooperative web servers. *Computer Networks*, 31(11–16):1215–1229.

Banga, G., Druschel, P., and Mogul, J. C. (1998). Better operating system features for faster network servers. *Performance Evaluation Review*, 26(3):23–30.

Barford, P. and Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. *SIGMETRICS Performance Evaluation Review*, 26(1):151–160.

Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. (1999). Web caching and Zipf-like distributions: Evidence and implications. In *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM '99)*, volume 1, pages 126–134.

Brisco, T. (1995). *DNS Support for Load Balancing. RFC 1794*. Internet Engineering Task Force.

Cardellini, V. (2003). Request redirection algorithms for distributed web systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):355–368.

Cardellini, V., Casalicchio, E., Colajanni, M., and Yu, P. S. (2002). The state of the art in locally distributed Web-server systems. *ACM Computing Surveys*, 34(2):263–311.

Casalicchio, E. and Colajanni, M. (2001). A client-aware dispatching algorithm for web clusters providing multiple services. In *Proceedings of the tenth international conference on World Wide Web*, pages 535–544.

Colajanni, M., Yu, P. S., and Dias, D. M. (1998). Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):585–600.

Devlin, B., Gray, J., Laing, B., and Spix, G. (1999). Scalability terminology: Farms, clones, partitions, and packs: RACS and RAPS. Technical Report MS-TR-99-85, Microsoft Research.

Garcia, J. D., Carretero, J., Garcia, F., Calderon, A., Fernandez, J., and Singh, D. E. (2006a). On the reliability of web clusters with partial replication of contents. In *First International Conference on Availability, Reliability and Security, 2006. ARES 2006.*, pages 617–624.

Garcia, J. D., Carretero, J., Garcia, F., Fernandez, J., Calderon, A., and Singh., D. E. (2006b). A quantitative justification to partial replication of web contents. In *International Conference on Computational Science and its Applications*, volume 3983 of *Lecture Notes in Computer Science*, pages 1136–1145.

Garcia, J. D., Carretero, J., Prez, J. M., Garcia, F., and Fernandez, J. (2003). A distributed web switch for partially replicated contents. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)*, volume VIII, pages 1–6, Orlando, FL, USA.

Kwan, T. T., McGrath, R. E., and Reed, D. A. (1995). NCSA's world wide web server: design and performance. *IEEE Computer*, 28(11):68–74.

Li, Q. and Moon, B. (2001). Distributed cooperative Apache Web server. In *Proceedings of the tenth international conference on World Wide Web*, pages 555–564.

Luo, M.-Y., Tseng, C.-W., and Yang, C.-S. (2002). URL formalization: An efficient technique to speedup content-aware switching. *IEEE Communications Letters*, 6(12):553–555.

Maltz, D. A. and Bhagwat, P. (1999). TCP splice for application layer proxy performance. *Journal of High Speed Networks*, 8(3):225–240.

Pai, V. S., Aron, M., Banga, G., Svendsen, M., Zwaenepoel, P. D. W., and Nahum, E. (1998). Locality-aware request distribution in cluster-based network servers. *ACM SIGPLAN Notices*, 33(11):205–216.

Pai, V. S., Druschel, P., and Zwaenepoel, W. (1999). Flash: An efficient and portable Web server. In *Proceedings of the USENIX 1999 Annual Technical Conference*, pages 199–212.

Pai, V. S., Druschel, P., and Zwaenepoel, W. (2000). IO-Lite: A unified I/O buffering and caching system. *ACM Transactions on Computer Systems*, 18(1):37–66.

Schroeder, T., Goddard, S., and Ramamurthy, B. (2000). Scalable web server clustering technologies. *IEEE Network*, 14(3):38–45.

Shukla, A., Li, L., Ward, A. S. P., and Brecht, T. (2004). Evaluating the performance of user-space and kernel-space web servers. In *Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research*, pages 189–201.

Tse, S. S. H. (2005). Approximate algorithms for document placement in distributed web servers. *Transactions on Parallel and Distributed Systems*, 16(6):489–496.

Vaidya, S. and Christensen, K. J. (2001). A single system image server cluster using duplicated MAC and IP addresses. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks (LCN 2001)*, pages 206–214.

Zhuo, L., Wang, C.-L., and Lau, F. C. M. (2003). Document replication and distribution in extensible geographically distributed web servers. *Journal of Parallel and Distributed Computing*, 63(10):927–944.