Generalization of the Fast Consistency Algorithm to a Grid with Multiple High Demand Zones

Jesús Acosta-Elias and Leandro Navarro-Moldes
Polytechnic University of Catalonia, Spain[1]
{jacosta, leandro}@ac.upc.es

Abstract. One of the main challenges of grid systems of large scale and data intensive is that of providing high availability and performance, in spite of the unreliability and delay occasioned by the size of Internet. Replication enables us to meet such a challenge with success. In the context of weak consistency, the fast consistency algorithm prioritizes replicas with high demand. Nevertheless, the fast consistency algorithm only works well in a single zone of high demand, whereas in multiple high demand zones its performance is poor. In this paper, we propose an algorithm chosen according to demand, whereby the replicas in each zone of high demand select leader replicas that subsequently construct a logical topology, linking all the replicas together. In this way, changes are able to reach all the high demand replicas without the low demand zones forming a barrier to prevent this from happening.

1 Introduction

Replication in a grid system should provide scalability [20], apart from increasing availability and performance – factors of great importance when working on Internet scale. Replicas in a grid system are made up of servers with the same content and providing the same service. If the system can withstand failures in both the links and the servers, availability increases. If a server cannot be reached by a group of users, these users are able to contact another server who can be reached at that time.

If the latency produced in the links is decreased, performance improves. Users will be able to contact the nearest server, thus avoiding having gone through links and routers in order to reach more distant servers. A further advantage is found in the saving of bandwidth, which is a resource in low supply. Considerable improvement in performance is also achieved, since there are many servers with the same data and services that can all attend user's requests simultaneously.

Replication algorithms have traditionally been classified into strong and weak consistency. The strong consistency algorithm ensures that all the replicas have exactly the same content (synchronous systems) before any transaction is carried out, where in an unreliable network like Internet, with a large amount of replicas, latency can become high so that it becomes impossible for using the system. The strong consistency

[1] This work has been partially supported by the Mexican Ministry of Education under contract PROMEP-57, the Spanish MCYT project COSACO (TIC2000-1054), and the Catnet EU project.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2658, pp. 275–284, 2003.

algorithm [5, 8, 9] is suitable for systems with few replicas, and on a reliable network where a large amount of bandwidth is available.

However, weak consistency algorithms, i.e. Golding's Refdbms [12], Thesis's Adya [4], Usenet [14, 22], Coda [23], Bayou [21], Ficus [13], Grapevine [24], generate very little traffic, low latency, and are more scalable. They do not sacrifice either availability or reply time in order to guarantee strong consistency, but only need to ensure that the replicas eventually converge to a consistent state in a finite, but not bounded, period of time. They are very useful in systems where it is not necessary for all the replicas to be totally consistent for carrying out transactions (systems that withstand a certain degree of asynchrony).

Golding's research into weak consistency prompted us to develop the "Fast Consistency" algorithm [1], which prioritizes the high demand replicas, and in which we see that with very few additional signaling bytes considerable improvement is obtained, since with a low number of anti-entropy sessions it is possible to deliver consistent content to a greater number of clients. Nevertheless, in [2] it is demonstrated that this algorithm has a very good performance in a zone with one high demand region, whereas in multiple regions of high demand its performance becomes poor due to the formation of islands of locally consistent replicas. To tackle this problem successfully, we propose a combined mechanism for converting multiple zones of high demand into a single zone, thus obtaining the best possible performance from the fast consistency algorithm.

2 The Model

The model of our distributed grid system consists of a number of N nodes that communicate via message passing. By simplicity we assume a fully replicated system, i.e, all nodes must have exactly the same content. But in real applications, while all nodes may want to be notified of changes, only a set of nodes may decide to keep a replica.

Every node is a server that gives services to local clients. Clients make requests to a server, and every requested service is a "read" operation, a "write" operation, or both. When a client invokes a "write" operation in a server, this operation (change) must be propagated to all servers (replicas) in order to guarantee the consistency of the replicas. An update is a message that carries a "write" operation to the replica in other neighboring nodes.

In this model, the demand of a server is measured as the number of service requests by their clients per time unit.

3 The Problem

The problem consists in that when there exist multiple zones of high demand, surrounded by zones of low demand, the propagation of the changes by means of the fast consistency algorithm is slowed up by the zones of low demand. In other words, the fast consistency algorithm is unable to carry the changes across the zones of low demand to the zones of high demand, since it has no overall knowledge of where the zones with high demand replicas are located. Each node that carries into effect both

the weak consistency algorithm and the fast consistency algorithm is aware only of its immediate neighbors; that is, all those replicas at distance one.

The existence of low demand regions, such as that we can see in Fig 1, zone II, and in which the updates arrive at a relatively low speed, gives rise to areas that the fast consistency algorithm finds difficult to overcome. It also causes the high demand regions (Fig. 1, zone I) to be isolated one from the other, and therefore their content undergoes delay before being mutually consistent.

These highly consistent regions surrounded by low consistency zones appear as islands formed by locally consistent replicas.

To illustrate this, a change produced in A, for example, quickly reaches B, but rises until C at a relatively slow speed, then falls rapidly to D. Ideally, the change produced in A should reach D just as rapidly as B, but in fact this is not the case since Zone II stands in the way.

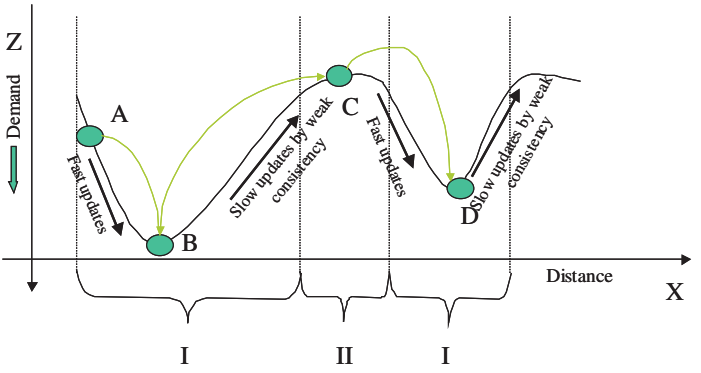

Fig. 1. Shows a cross section of the distribution of nodes. The demand is on the Z axis, and on the X axis we have the distribution in the X-Y plane, where Y = k. The nodes are shown in such a way that those with the most demand are towards the bottom, and those with the least demand are found towards the top. Two different kinds of regions can be seen, valleys (I) and mountains (II).

4 Proposal

Using an election algorithm [16, 6, 11], the replicas in each region of high demand will choose a leader node. The leader node in each region of high demand (island) must be one of the nodes of greater demand (most requests) in the zone. The changes brought about within the zone by the fast consistency algorithm reach the nodes of greater demand in each zone. This algorithm ensures that the changes reach these zones in very few sessions.

When every island has a representative, a superset of representatives is formed, which make up a logical topology connecting all the islands together. Its main task will be to take all the changes brought about in each region to the rest of the regions, without having to cross the zones of low demand.

4.1 Election Algorithm

A choice begins at a configuration in which all the nodes are in the same state and reach a configuration exactly where one node is the leader. It is necessary to make a choice; either when a centralized algorithm is put into effect in a distributed system, or after a system crashes and it is not known which nodes are functioning.

Definition of the election algorithm: Each node has the same local algorithm. The algorithm is decentralized when it is able to start an execution by an arbitrary non-empty subset of nodes.

It is called a leader network, or it is said to contain a leader if there is exactly one node, which knows that it is "the leader". The availability of a leader can be exploited by providing it with a distinguished local algorithm, while all other nodes execute the same local algorithm (which is different from the leader algorithm).

One may now deduce that the model that adapts to the problem of finding the node with most demand is that belonging to a leader network. Nevertheless, this algorithm is more general. Our problem is more specific, since the leader node is the one with most demand and is found at the bottom of a high demand zone.

All the nodes in a zone of high demand cooperate together to select the node, which will carry out the processes of the leader node.

4.2 The Algorithm for Electing the Leader Node on a Zone of High Demand

For the development of this algorithm, we consider a peer-to-peer network with N nodes connected to each other via E links and with a distinct and finite weight (demand) on each one of the nodes. Each node executes the same local algorithm, which consists in first sending messages (announcing its demand) to its neighbors via the corresponding (adjacent) links, awaiting the arrival of the messages (neighboring demand) and processing them. The messages are transmitted in all directions and arrive after an unpredictable but finite delay.

Each node at a random time (fig. 3) will cast its vote for the neighboring node having the greatest demand, and will send it a message notifying it that the vote has been cast. Each vote is unique and unrepeated; it has the ID of the node casting it, a time stamp, and a time to live (in real applications it is an empirical number) necessary for avoiding loops or for preventing the vote from circulating infinitely around the network.

Each node that receives a vote passes the vote on to whichever of its neighbors has the largest number of requests, and so on successively, until after an unbounded but finite period of time the majority of votes cast on an island (high demand zone) have only one node, which will be the node selected (the leader). It is not possible to ensure that all the votes of the nodes on an island reach the node of greatest demand, since the number of nodes that make up a zone of high demand is not known. Neither do we know how many votes are still traveling without having arrived at the node of greatest demand. However, it is possible to ensure that the votes in a high demand zone will not travel to other zones, since only replicas of higher demand are propagated, and never toward the zones of lower demand. In order for a node to take on the

role of a leader node, it is sufficient that, after a period of time, the number of accumulated votes is different from zero.

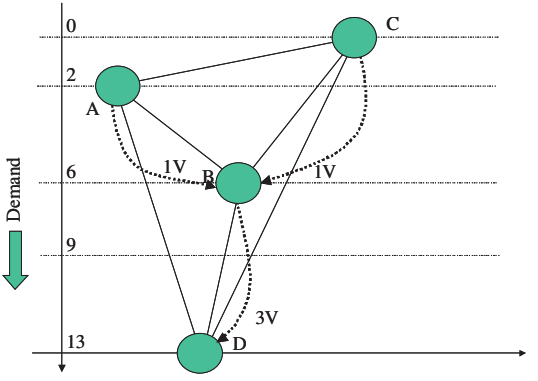

Fig 2. An example of a demand based leader election algorithm.

Table 1. Accumulated votes in the nodes of figure 2.

Node	Votes
A	0
B	0
C	0
D	3

Informally speaking this algorithm works in the following way (fig. 2):

- Node C votes for its neighbor having most demand (B) and sends it its vote.
- Node B receives a vote from C, and since B has a neighbor with greater demand, it sends C's vote to this neighbor (D).
- Node D receives C's vote from B. D looks to see if it has a neighbor with greater demand. Since it does not, it keeps the vote. So D now has one vote.
- Node A votes for its neighbor with greatest demand (B), and sends it its vote.
- Node B receives a vote from A, and as B has a neighbor with greater demand, it sends A's vote to this neighbor (D).
- Node D receives A's vote from B. D looks to see if it has a neighbor with greater demand. It does not, so it keeps the vote. D now has two votes
- Node B votes for its neighbor having greatest demand (D), and sends it its vote.
- Node D receives B's vote. D looks to see if it has a neighbor with greater demand. It does not, so it keeps the vote. D now has three votes.

If node D sees no nodes with more demand than itself, it does not send the votes it has received to any other node; neither does it cast its own vote. Therefore it retains all the votes, and thus becomes the leader node in this high demand zone.

Every time it is necessary to construct the superset of leaders, it is enough for each node having a number of votes greater than zero to announce itself.

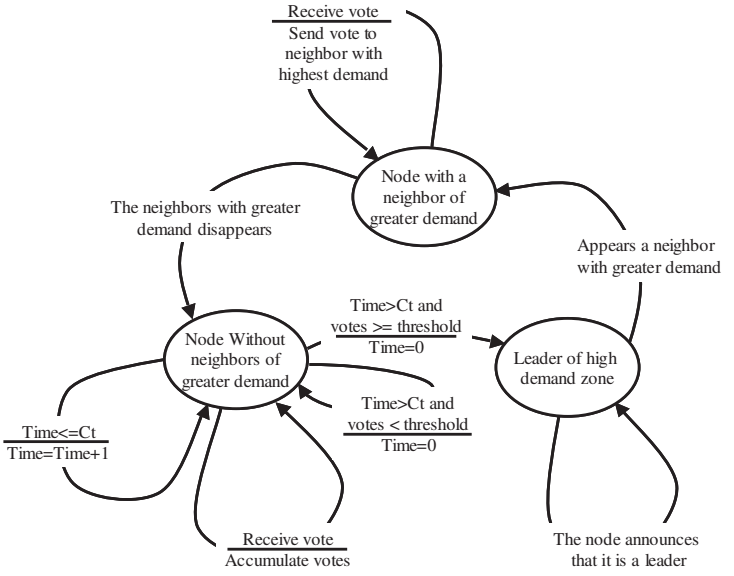

Fig.3. The state machine of leader election algorithm.

4.3 The Building of the Leader Network

The node that knows it is the leader now has the task of finding other leader nodes, if they exist. It must also announce itself so that other nodes become aware of its existence. The mechanism by which a leader node announces itself is by sending a message as part of the weak consistency protocol. This protocol ensures that the message arrives to all the nodes in a finite, but unbounded, period of time, and therefore to the leader nodes as well, assuming they exist.

When a leader node receives a message from another leader node, it keeps the ID of the node sending the message in a table. Each leader node has a table containing the data of the other leader nodes that know of its existence. This table is replicated in each leader node and is reconstructed dynamically. The ID of a leader node is included in the table on arrival of a message of announcement. It is not necessary to remove a node from the table of leaders because the table is dynamically reconstructed periodically, the period of time being at least equal to the time (expressed in sessions) necessary for the message to cross the entire network of replicas.

5 Validation

In order to validate our proposal, a simulator based on NS2 [19] has been built. We simulate the behavior of the algorithms on a grid network with synthetic demand. Our fast consistency algorithm has been tested in [1] using a generator of random representative Internet topologies [10,17,18], now in this paper we use as scenarios for apply-

ing our algorithms, surfaces of 17*17 nodes on which the different levels, representing the demands, are synthetically generated by the diamond-square algorithm, which is a classic algorithm for generating fractal surfaces that resemble landscapes with scaling properties; that is to say, self-similar. In this way, we achieve a scenario sufficiently general to ensure that the results obtained in the simulations do not depend on the particular or local conditions of a specific scenario. To reduce the effects of randomness, and to prevent the results from depending on the characteristics of a particular fractal surface, one hundred different scenarios have been generated, on each one of which a thousand experiments have been carried out.

5.1 Performance Metric

The purpose of the "fast consistency" algorithm with high demand zones interconnected is to improve the performance of the weak consistency algorithms, with particular emphasis on increasing the speed with which these algorithms carry the changes to the zones of greatest demand, so that a greater number of clients may have access to fresh content in a shorter period of time. It is for that reason that our experiments are centered on measuring these speeds. The performance (speed) is measured in terms of the anti-entropy sessions needed for all the zones to receive the messages with the changes generated en the rest of the nodes that make up the grid.

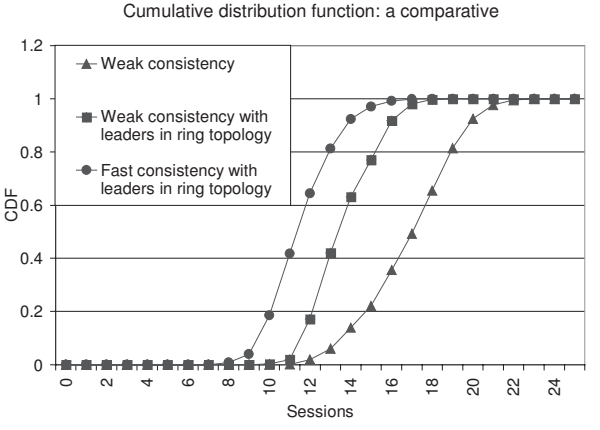

Fig. 4

5.2 Simulation Results

The leader nodes having a number of votes greater than or equal to a threshold (i.e. average of all node leaders) are selected and interconnected in a ring topology, which joins together the zones of high demand. In this topology, each node sees the same

network diameter. The results can be seen in Figure 4(CDF); we see that Fast Consistency with leaders interconnected has a better performance than Weak Consistency, and also the effects of the change on the topology when the leader nodes are connected in a ring topology.

6 Related Work

Work whose aim is to adapt replication systems to clients' needs, in TACT [25] for example, propose that the consistency between replicas vary in a continuous way between weak and strong. Three metrics are used to limit consistency – numerical error, order error, and staleness. In TACT, it is the client who specifies the level of consistency required. This level may take any value between weak and strong. With fast consistency, however, updated content is carried to the high demand replicas without incurring the huge costs involved with the strong consistency.

In Fluid Replication [7], clients can create replicas dynamically, whenever and wherever they are necessary. When a client sees that performance is low, either because of increasing demand on the network, or because of client mobility, he can create replicas on a waystation, which is a node on which replicas can be created. This system seeks to provide replicas when and where they are required. Our proposal concerns algorithms for keeping replicas consistent. Our algorithms can be used to complement the fluid replication system. In order that replicas in fluid replication permanently updated by fast consistency could be created wherever there was a great concentration of clients. It is in such cases where our algorithm could act as an excellent complement to fluid replication.

In [15], S. Krishnamurthy and others put forward a dynamic selection replica algorithm based on the historic performance of a set of replicas, and which is capable of dynamically selecting replicas having all the service quality requirements a client might request. For example, a client who needs a service that responds to requests within a specific time expresses these requirements as a service quality specification. The client can specify his QoS requirements at the beginning or during execution. The specification includes the name of the service and the expected response time. A scheduler intercepts clients' requests, estimates the response time of the different replicas offering the service the client is asking for, and selects the subset of replicas that have what the client requires. We propose is giving preferential treatment to large client subsets. Furthermore, the dynamic replica selection algorithm has been tested on AquA; a middleware based on CORBA in local area networks, although it works very well in large scale distributed systems such as Internet.

7 Conclusions and Future Work

The main challenge to distributed systems on Internet scale is that of maintaining availability and performance, in spite of the low reliability and the delay associated with large size broad networks. To meet this challenge successfully, the most valuable tool is replication. However, it is not a magical solution; it involves the problem of

keeping replicas consistent. Weak consistency is an algorithm that maintains replicas consistent, sacrificing data freshness in a controlled way in order to improve availability and performance. In the context of weak consistency, the fast consistency algorithm prioritizes replicas with high demand in such a way that a large number of clients receive fresh content. This algorithm gives high performance in one zone of high demand, but in multiple zones this performance becomes poor. To improve this poor performance, we propose an election algorithm based on demand. After an interval of time t, the replica that accumulates a great number of votes in a zone of high demand assumes the role of leader in that zone. The leaders in each zone together make up a logical hierarchy [3] whose task is to convey the changes made in each zone to the high demand replicas, preventing the zones of low demand from forming a barrier to the fast consistency algorithm. To validate this proposal we have constructed a simulator on Network Simulator [19]; we have feed it with a set of replicas in which there exist multiple zones of high demand, connecting these zones together by means of a logical topology. The result is a considerable improvement in the algorithm performance compared with the same topology without the interconnected high demand zones. This allows us to conclude that with very few additional bytes in signaling, and increasing the complexity of the fast consistency algorithm a little – since periodic election must be carried out so that the hierarchy of leader replicas corresponds to the dynamic state of the system – demand is changeable, and the leader nodes and the links between them must be kept up to date. In this work we have not considered the effect of the local minimums, which can slow down fast consistency and/or create false leaders. It may be necessary for the replicas to be sensitive to field effects, as it occurs in the relativity theory of gravity, and thus the replicas would have information beyond their neighbors lying at distance one.

References

1. Jesús Acosta Elias, Leandro Navarro Moldes, “A Demand Based Algorithm for Rapid Updating of Replicas ”, IEEE Workshop on Resource Sharing in Massively Distributed Systems (RESH’02), July 2002.
2. Jesús Acosta Elias, Leandro Navarro Moldes, “Behaviour of the fast consistency algorithm in the set of replicas with multiple zones with high demand”, Symposium in Informatics and Telecommunications, SIT2002.
3. N. Adly, M. Nagi and J. Bacon. 'A Hierarchical Asynchronous Replication Protocol for Large Scale Systems'. Proc. of the IEEE Workshop on Parallel and Distributed Systems, Princeton, New Jersey, October 1993, pp.152-157.
4. Adya, “Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions”, PhD thesis MIT, Department of Electrical Engineering and Computer Science, March 1999.
5. K. P. Birman, “The process group approach to reliable distributed computing”, Communications of ACM, December 1993/Vol. 36, No. 12
6. Ernest Chang, Rosemary Roberts, “An improved algorithm for decentralized extrema-finding in circular configurations” of processes Communications of the ACM May 1979 Volume 22 Issue 5

7. L. P. Cox, and Noble, B. D. 2001."Fast Reconciliations in Fluid Replications", Int. Conf. On Dist. Comp. Syst. (ICDCS) (April 2001). http://mobility.eecs.umich.edu/papers/icdcs01.pdf
8. D. J. Dietterich, "DEC data distributor: for data replication and data warehousing". In Int. Conf. On Management of data, pp 468, ACM, May 1994.
9. V. Duvvuri, P. Shenoy and R. Tewari, "Adaptative Leases: A Strong Consistency Mechanism for the World Wide Web", IEEE INFOCOM 2000, p 834-843.
10. M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology", ACM SIGCOMM, Cambridge, MA, September 1999
11. R. G. Gallager, P. A. Humblet, P. M. Spira, "A Distributed Algorithm for Minimum-Weight Spanning Trees" ACM Transactions on Programming Languages and Systems (TOPLAS) January 1983 Volume 5 Issue 1
12. R. A. Golding, "Weak-Consistency Group Communication and Membership", PhD thesis, University of California, Santa Cruz, Computer and Information Sciences Technical Report UCSC-CRL-92-52, December 1992.
13. R. Guy, J. Heidemann, W. Mak, T. Page, Jr., G. Popek and D. Rothmeier. Implementation of the Ficus Replicated File Sys. Proc. USENIX Conf, June 1990.
14. Brian Kantor, Phil Lapsley RFC0977, http://www.ietf.org/rfc/rfc0977.txt, 1986
15. S. Krishnamurtyhy, W. Sanders, and M. Cukier, "A Dynamic Replica Selection Algorithm for Tolerating Timing Faults", In Proc. Of the The International Conference on Dependable Systems and Networks, pages 107-116, July 2001.
16. G. LeLann, "*Distributed systems - towards a formal approach*," in Information Processing 77, B. Gilchrist, Ed. North-Holland, 1977.
17. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal Topology Generation from a User's Perspective",
18. Medina, I. Matta, and J. Byers, "On the Origin of Power Laws in Internet Topologies", ACM Computer Communication Review, p. 160-163, April 2000.
19. The Network Simulator: http://www.isi.edu/nsnam/ns/
20. C. Neuman, "Scale in Distributed Systems. In Readings in Distributed Computing Systems", IEEE Computer Society Press, 1994
21. K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and Demers, "Flexible Update Propagation for Weakly Consistent Replication", Proc. of the 16th ACM Symp. on Op. Syst. Prin. (SOSP-16), S. Malo, France, Oct. 5-8,97, p. 288-301.
22. Yasushi Saito, Jeffrey C. Mogul, and B. Verghese. A Usenet Performance Study, http://www.research.digital.com/wrl/projects/newsbench/usenet.ps. Nov 98."
23. M. Satyanarayanan, Scalable, Secure, and Highly Available Distributed File IEEE Computer May 1990, Vol. 23, No. 5
24. Michel D. Schroeder, Andrew D. Birrel, and Roger M. Needham, Experience with Grapevine: The Growth of a Distributed System, ACM Transactions of Computer Systems, Vol. 2, No. 1, February 1984, Pages 3-23.
25. Haifeng Yu, Amin Vahdat, "Desing and Evaluation of a Continuous Consistency Model for Replicated Services", Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI), October 2000.