Title:

Behavior of Fast Consistency Algorithms in Large Decentralized Systems

Authors and affiliations:
Jesús Acosta-Elias, Leandro Navarro
Polytechnic University of Catalonia, Spain
{jacosta, leandro}@ac.upc.es

Contact author:
Leandro Navarro
UPC, Dep. AC, D6-105
Jordi Girona, 1-3
08034 Barcelona
Spain
Email: leandro@ac.upc.es

Brief abstract:

Information replication is a major technique for improving the efficiency and autonomy of many distributed applications. Weak consistency algorithms allow us to propagate changes in an arbitrary, changing storage network in a self-organizing way. The weak consistency algorithms generate very little traffic overhead; they have low latency and are scalable, in addition to being fault tolerant. In this paper we evaluate our own weak consistency algorithm, which is called the "Fast Consistency Algorithm", and whose main aim is the propagation of changes with preference for nodes and zones of the network which have greatest demand.

The algorithm has been simulated over ns-2, and evaluated with several topologies: linear, ring, random and grid, and several distributions of demand: constant, one region of high demand, two regions, and self-similar fractal mountains of demand.

The impulse response of the algorithm has been characterized: the time it takes for all the network to become consistent; i.e. measuring the speed with which each node receives all events introduced at t=0 in the rest of the nodes.

Employing, among other the economic concept of an utility function, we show that our "fast consistency" algorithm, interconnecting high demand zones by means of a logical topology, optimizes the distribution of changes by prioritizing the nodes with greatest demand, independently of demand distribution. In other words, it satisfies the greatest demand in the shortest amount of time.

Jesús Acosta-Elias is student: grant by the Mexican Ministry of Education (Secretaría de Educación Pública) under contract PROMEP-57.

Regular presentation

Behavior of Fast Consistency Algorithms in Large Decentralized Systems

Jesús Acosta-Elias, José Mitre, Leandro Navarro-Moldes
Polytechnic University of Catalonia, Spain
{jacosta, jmitre, leandro}@ac.upc.es

Abstract

Information replication is a major technique for improving the efficiency and autonomy of many distributed applications. Weak consistency algorithms allow us to propagate changes in an arbitrary, changing storage network in a self-organizing way. The weak consistency algorithms generate very little traffic overhead; they have low latency and are scalable, in addition to being fault tolerant. In this paper we evaluate our own weak consistency algorithm, which is called the "Fast Consistency Algorithm", and whose main aim is the propagation of changes with preference for nodes and zones of the network which have greatest demand.

The algorithm has been simulated over ns-2, and evaluated with several topologies: linear, ring, random and grid, and several distributions of demand: constant, one region of high demand, two regions, and self-similar fractal mountains of demand.

The impulse response of the algorithm has been characterized: the time it takes for all the network to become consistent; i.e. measuring the speed with which each node receives all events introduced at t=0 in the rest of the nodes.

Employing, among other the economic concept of an utility function, we show that our "fast consistency" algorithm, interconnecting high demand zones by means of a logical topology, optimizes the distribution of changes by prioritizing the nodes with greatest demand, independently of demand distribution. In other words, it satisfies the greatest demand in the shortest amount of time.

Keywords. replication, weak consistency, information dissemination.

1. Introduction

A growing number of Internet applications need to run on a changing and unreliable network environment with a very large number of clients. Peer-to-peer and grid based applications are examples of these, where equivalent software is running at each location. Collaborative and other data intensive applications strongly depend on efficient access to reasonably up-to-date information. Replication is one way to provide service to clients with low delay response, high degree of availability and autonomy (independent of unexpected backbone delays or link failures), and good scalability. This work is based on a model of service composed by a number of replicas running on hosts (nodes) at different locations.

A replica is a host which provides exactly the same services as the principal host. In this paper we will use the terms server and replica in the same sense.

A non-trivial problem is how to keep all the replicas consistent, with the same content, when changes in information are introduced. The distribution of the changes to all the nodes (replica updating) can be dealt with using various algorithms, but several issues have to be considered:

– Consistency: there are strong consistency algorithms, and weak consistency algorithms.

Strong consistency algorithms [10,15,28] are suitable for synchronous systems with a small number of replicas, where it must be guaranteed that all the replicas are in a consistent state (i.e. all the nodes possess exactly the same content) before any transaction can be carried out. Therefore they are costly, non-scalable on unreliable networks, generating considerable latency and a big amount of traffic.

By contrast, weak consistency algorithms [1,23,16] generate very little traffic, have low latency, and are more scalable. They do not sacrifice either availability or response time in order to guarantee strong consistency, but only need to ensure that the nodes 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 nodes to be totally consistent in order to carry out transactions (systems that withstand a certain degree of asynchrony).

– Distribution of Demand: we cannot assume that demand is the same in all locations. Demand is dynamic: there may be hot spots of demand at some locations, meanwhile somewhere else demand could be several orders of magnitude smaller.

In this paper we propose and evaluate "Fast Consistency" (FC), a weak consistency algorithm for the dissemination of changes considering demand. In this scenario, each network node provides service to a group of subscribers, and nodes are only required to know a few neighbour nodes (autonomy and self-organisation). FC gives priority to zones of higher demand, thus satisfying the greatest demand in the shortest amount of time: more users will perceive faster updates with the same number of messages.

Replication with weak consistency is not new; it was already used as a mechanism in Grapevine [21] for providing availability and increasing performance. With the weak consistency algorithms, i.e. Golding's Refdbms[23], Adya's work [1], Usenet [5,30], Coda [18], Bayou [16], Ficus [25], Grapevine [201], each node from time to time chooses a neighbour to start an update session. In an update session two nodes mutually update their contents. At the end of the session both nodes will have the same content. These are called anti-entropy sessions: It is called an anti-entropy session because in each session between nodes, the total entropy in the network is reduced. In this paper it will be referred to simply as a "session".

The usual metric principle to evaluate weak consistency algorithms is the amount of sessions necessary for a change brought about in a node to be propagated to all the others.

Previous research into weak consistency prompted us to develop the "Fast Consistency" (FC) algorithm [12]. We have found considerable improvement with the exchange of very little additional signalling information: with a low number of anti-entropy sessions it is possible to deliver consistent content to a greater number of clients (satisfying most demand in less time).

In [13] we already 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 advantage is reduced due to the formation of islands of locally consistent replicas. To tackle this problem successfully, we proposed in [14] a combined mechanism for converting multiple zones of high demand into a single zone: interconnecting the leaders of every zone of high demand, thus obtaining the best possible performance from the fast consistency algorithm.

This paper presents a study, by means of simulation, of our "fast consistency" algorithm over several topologies and distributions of demand. Given that the worst case demand has a combination of high and low demand zones, the value of demand could be viewed as a landscape consisting of mountains and valleys of demand. For this purpose, we have developed a random demand generator with self-similar characteristics, in the form of mountains and valleys, using the diamond-square algorithm [2] from computer graphics.

To evaluate the performance of the algorithm presented in this paper, a fast and weak consistency algorithm simulator has been constructed, over Network Simulator 2 (ns-2) [27].

To take into account the demand of clients at every node we use additional metrics: the speed of demand satisfaction (the rate of demand satisfied with consistent information), an utility function (based on economic theory). We conclude that FC improves the distribution of changes by prioritizing nodes with greatest demand, rather independently of demand distribution and topology. In other words, our algorithm satisfies the greatest demand in the shortest amount of time, while sending the same amount of messages (better value at the same cost).

The rest of the paper is organized as follows: Section 2 describes our system model. Section 3 describes the "fast consistency" algorithm. In section 4 we explain the methodology of simulation of our algorithms in terms of network topology, demand workload and performance metrics. In section 5 we discuss the simulation results for several cases. Section 6 describes how the "fast consistency" algorithm relates to other proposals where replication adapts somehow to the demand. The paper concludes in section 7.

2. System model

In a typical collaborative application or data-intensive application, participants are clustered at different locations with one local server that knows something (demand in our case) about some neighbors but not all servers. A simple value of demand for every server corresponds here to the number of their local clients. Each circle may correspond to a region of good and reliable connectivity (e.g. an office, campus, building, school).

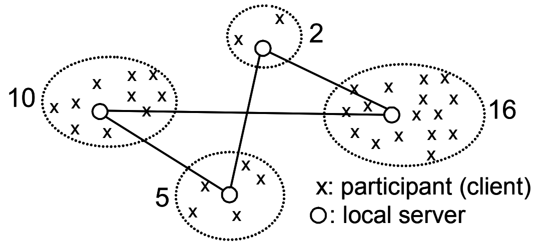

Figure 1. In many collaborative and data-intensive applications, it may be beneficial to group client around several main servers, nodes or 'principals'.

The model of our distributed system consists of a number of N nodes (principals) that communicate via message passing. We assume a fully replicated system, i.e., all nodes must have exactly the same content.

Every node is a server that gives services to a number of local clients. Clients make requests to a server, and every request 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 or simply the number of clients "subscribed to".

We are interested in dynamic application scenarios requiring:

– self-organisation: nodes only need to know about a few neighbours, and the FC algorithm eventually propagates changes to any distant node in a few protocol sessions (a few time) without any required order or topologic knowledge,
– fault-tolerance: changes propagate randomly through the network circumventing failed nodes or links,
– adaptation to changes of demand: FC automatically adjust to changes in demand. It gives priority to nodes with greatest demand, but in any case it randomly interacts with all neighbours.
– scalability: every node only needs to know a few neighbouring nodes, almost independent of the total number of nodes.
– autonomy: since a community of clients is served by a single node (that could be implemented with one or several machines for higher availability) with a copy of the relevant data, clients are independent of the rest.

The FC algorithm can be implemented as a daemon that interacts with a local server application. The daemon is responsible for propagating data, and thus maintaining consistent the external and local worlds. The daemon needs to be informed of the local interests (subscription to data), the value of local demand, and any new or changed data of external interest. It provides the local application with updates from the rest of the network. The internal state information consists on a vector of demand (local and neighbour nodes), and data (state vector of data), At random time it selects with random neighbour peer nodes (with preference for nodes with greater demand) to exchange new data.

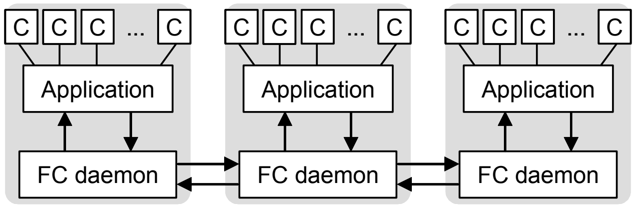

Figure 2. Structure of a distributed application with FC replication

Examples of applications that benefit from the use of the FC algorithms are:

– Collaborative: peer-to-peer applications that provide autonomous and decentralized access to data produced by a group of authors and readers working at different locations. Since the documents of interest for every participant are almost fresh and stored locally or very close, access for frequent read or modification can be extremely efficient, independent of external failures, and since updates come through a random path, interests and activity are kept private. The propagation of changes beyond the local node does not imply any delay, configuration or intervention for humans.
– Data-intensive: grid based applications requiring intensive access to large and changing data sets (e.g. financial, scientific) may express interest and have an asynchronously but very inexpensive fresh information locally accessible data. This is specially important in applications where the response time mainly depends on the access time to storage (e.g. Datagrid).

Many other examples may come from asynchronous applications requiring intensive access to certain mutable data, requiring fast and selective replication of information on large and dynamic networks of autonomous agents with incomplete information, and clients or participants with varying degrees of interest, scattered or clustered in different locations.

3. The Fast Consistency algorithm

The FC algorithm describes how changes propagate among network nodes with different value of demand. The following section describes an extended TSAE weak consistency algorithm with a "fast update" step for faster propagation of changes to nodes of higher demand. Section 3.2 describes the leader interconnection algorithm, an improvement of the first algorithm to improve performance when there are several regions of higher demand surrounded by regions of lower demand. Section 4 and 5 evaluate the behaviour of these algorithms with different interconnection topologies and distributions of demand.

3.1 The basic FC algorithm

In a network with an arbitrary large number of nodes N, every node n knows his local demand and the demand on t neighbours: $n.demand$ or d_n, and $n.neighbours[n_i]=\{ d_0, d_1, d_n, d_t, \}$. Demand at node n: d_n, in our model and simulations it has been defined simply as the number of clients that node n provides service. The value of t is typically in the range of $1..log(N)$.

Every message m can be identified by a MessageId and a Timestamp: $m = \{ m.id, m.tstmp, m.data \}$

Every node n has a summary vector of the history of messages it has received: $n.SV[\,]$

Figure 3 describes the protocol with an example among three neighbour nodes, where n has a new message m and $n''.demand > (n.demand, n'.demand)$, with all values of demand observed at the nodes at the same time. n randomly initiates a session of exchange of summary vectors with n' and n' immediately sends a fast update with n'' because it has greater demand.

A fast update of message m is $FU(m)=\{m.id, m.tstmp\}$, and $sizeof(m) >> sizeof(SV) >> (sizeof(FU), ...)$.

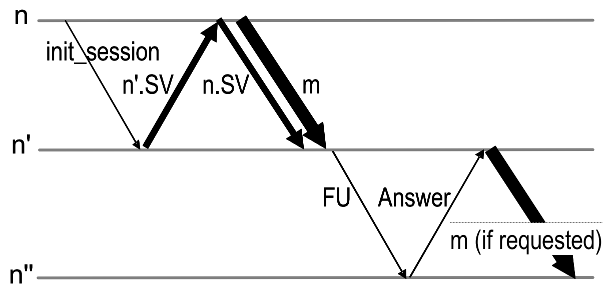

Figure 3. An anti-entropy session followed by a fast-update notification to a node with higher demand.

```
n  | n' | n''
-----------------------------------------------------------------
after n.wait(random time)
n'=n.select_neighbour(n.neigbours[ ],HIGHEST_DEMAND)
n.send_init_session(n') // message sent to n'
-----------------------------------------------------------------
     when n'.receive_init_session:
       n'.send_SV(n,n'.SV[ ]) //SV: summary vector
-----------------------------------------------------------------
when n.receive_SV:
  n.send(n',n.SV[ ])
//Calculate diffV==messages n' not yen received
diffV=n.compareSV(n.SV[ ], n'.SV[ ])
foreach message m in diffV:
  n.send(n', m) //send message m to n'
-----------------------------------------------------------------
     When n'.receive(n.SV[ ]):
     diffV[ ]=n.compareSV(n'.SV[ ], n.SV[ ])
     foreach message m in diffV[ ]:
       n'.send(n, m)
-----------------------------------------------------------------
     when n'.receive(m):
       n'.process(m) //and update n'.SV[ ]
     foreach neighbour e in n'.neighbours[ ]
       where n'.neighbours[e].demand > n'.demand:
         FU=new fast_update(m.id, m.tstmp)
     n'.send_fast_update(e,FU)
-----------------------------------------------------------------
               when n''.receive_fast_update(FU):
               if n''.check_history(FU) == UNKNOWN):
                 n''.request(FU.id)
               else n''.reject(FU.id)
-----------------------------------------------------------------
     when n'.receive(FU.id): //if requested
       n'.send(n'', m) //send message m to n''
```

Figure 4. The pseudo-code for the FC algorithm instantiated for figure 3.

3.2 Leader interconnection

In order to improve performance when there are several regions of higher demand surrounded by regions of lower demand, the nodes in high demand zones choose a leader node by means of a decentralized voting algorithm [14, 7, 24, 8]: each node casts a vote for its neighbour with greatest demand. At the end of the voting process there exists a set of nodes that have accumulated votes. However, not all the nodes possessing votes can be considered leaders, since there will be nodes with more votes and others with very few votes. Thus, from this subset of nodes, all those having fewer votes under a certain threshold will be discarded.

In this way we obtain a new set of nodes which represent the greatest zones of high demand, but as sometimes occurs in a democratic political election, the nodes with the largest number of votes are not necessarily those with greatest demand, although they may represent a high demand zone because they are the nodes in this zone with greatest demand.

These leader nodes establish connection among themselves in order to speed up the exchange of messages among all the zones of high demand. Therefore, the logical communication topology is slightly transformed into the equivalent of one zone of high demand, that is the most favourable situation for our algorithm.

4. Simulation methodology

To evaluate the performance of the fast consistency algorithm compared to the baseline TSAE algorithm [23], we simulate the behavior of the algorithms on different topologies (line, ring, random network, grid) and different demand distribution at nodes (constant, one zone with high demand, two zones, random, fractal).

In a simulation run each node originates a new message at t=0. The simulation measures the number of FC sessions required to reach a global consistent state: every node will receive the total number of messages in different number of sessions.

In this section, we discuss which network topologies and demand workloads have been selected for our simulations. We then describe the performance metrics that we use as a basis for comparing the algorithms in terms of how demand is satisfied in time.

In the simulation we have discarded the influence of network performance (latency, bandwidth, congestion) because the time required to send a message from one node to another was assumed to be negligible compared to the time between anti-entropy sessions.

4.1. Network Topology

Several simulations have been performed using various topologies. Initially a linear topology and a ring were used, in order to verify the correct behaviour of the simulator and to obtain fast and clarifying results. Afterwards the same experiments were repeated using a random topology generated by BRITE [19,20,30] which reflects many aspects of the actual Internet topology.

We have observed a certain degree of independence of the network organization: other experiments using different topologies and different number of nodes (linear, ring, random networks generated with BRITE) have shown similar results: the mean number of session to reach a global consistent state is more related to the network diameter than to the number of nodes.

Therefore, and in order to obtain realistic results respecting the limited computational power available with a simple topology, our scenario consists of a square-sectioned mesh of 17x17 nodes.

Given the observed dependence between mean number of sessions to reach a global consistent state and network diameter, results for our network with diameter=17 could be applicable to a network topology similar the Internet. Danesh et al. [4] claim that informed random address probing with TTL up to 15 discovers most of the network hosts, and the number of new hosts using probes with values of TTL between 15 and 30 does not grow significantly.

4.2. Demand Workload

The demand in our experiments corresponds with the number of clients who use a certain replica. In certain applications the definition of what demand is can be different. For instance, in a publish/subscribe collaborative application, demand would correspond to the number of subscribers to a certain section. In other applications demand could be calculated from the history of past requests.

We have considered several cases of demand:

– Constant: same demand for all network nodes. The FC algorithm with constant demand has exactly the same behavior as the TSAE algorithm. Several identical or equivalent results as those obtained for TSAE in [23] has helped to validate our simulator.
– One zone of high demand: demand is maximum on the centre of the topology and decreases towards the extremes. Here the FC algorithm works best,
– Two zones: the performance of the FC algorithm degrades as there are two regions of high demand separated by a region of low demand. The leader interconnection algorithm helps to obtain similar results as if it was one zone,
– random-fractal: an extreme but realistic case, where the distribution of demand has random mountains and valleys of demand. This model seems to be an extreme case with similar characteristics to the spatial distribution of demand on the Internet.

In recent works of Yook et al. [26], and in [3] Anukool et al. demonstrated a similar fractal dimension (≈ 1.5) of routers, ASes, and population density. The coincidence between the fractal dimension of the population and the Internet nodes (router and AS) is not unexpected: high population density implies higher demand for Internet services, resulting in higher router and domain density. The demand is generated by the Internet users. If the geographic location of Internet users have fractal properties, we can infer that the demand has the same fractal properties. Other important characteristic is the existence of high demand regions and large regions of low demand [11].

We use as scenarios for applying our algorithm, 100 random demand surfaces on which the different values of demands, are synthetically generated by the diamond-square algorithm [2], which is a classic algorithm for generating fractal surfaces that resemble landscapes with scaling properties or self-similar, and a fractal dimension between 2.0 and 3.0, which is a worst case respect the values of 1.5 for the Internet. Using 100 random fractal demand surfaces, 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, as we have verified comparing results.

The results may also be biased if the demand is calculated using simple random number generators, as we found experimentally. OpenSSL's RAND-pseudo-bytes function, a cryptographically secure pseudo-random number generator based on a cryptographic hash function, is used as a generator of "good" random numbers [22]. This function generates random bytes with a uniform distribution.

To reduce the effects of randomness, and to prevent the results from depending on the characteristics of a particular fractal surface, each experiment has been run 1000 times for every (100) random demand surface.

4.3. Performance Metric

The purpose of the "fast consistency" algorithm is to improve the performance of the weak consistency algorithms, with particular emphasis on increasing the speed with which these algorithms convey 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 centred on measuring these speeds.

At the beginning of a simulation, at t=0, a new message is generated in every network node. The experiment ends when each node receives all the messages. The performance (speed) is measured in terms of the number of anti-entropy sessions needed for all the zones to receive the messages originated at every replica (node).

If the number of users in each node of the network is used as a measure of demand, then a node with a high number of users which reaches a consistent state will benefit the user community more than another node with a low number of users which reaches the same state at the same time. The availability of up-to-date information on a distributed application will be higher if high demand nodes have higher priority than low demand ones.

Every simulation calculates the pair (d_i, c_i) for all nodes, where d_i is the demand at node i, and c_i is the time when node *i* has received all changes. This pair can be expressed by the *c(i,t)* function (an impulse function of value d_i):

$$(1) \quad c(i,t) = \begin{cases} d_i & : t = c_i \\ 0 \end{cases} \qquad C(t) = \sum_{i=0}^{N} c(i,t)$$

C(t) is the sum of demand for all nodes that have reached a consistent state at a certain time t.

In economic terms, we can define a utility function for each node u(i,t). It represents the value of demand satisfied with up-to-date information at time t (a step function of value d_i).

$$(2) \quad u(n_i,t) = \begin{cases} d_i & : t \geq c_i \\ 0 \end{cases} \qquad U(t) = \sum_{i=0}^{N} u(n_i,t)$$

U(t) is the sum of utility for all nodes that are consistent in time $\leq$ t.

U(t) expresses the satisfaction or benefit perceived by the community of users of our system.

At time t=0, all nodes of the network are in a non-consistent state: U(t)=0, and as time passes more and more nodes will reach a consistent state and thus they will contribute to the growth of U(t) with their local demand d_i. The faster U(t) grows, the better.

5. Simulation results

In this section, we evaluate the performance of the various parts of the algorithm on several topologies and demand workloads.

Every experiment shows the mean value after more than 1000 executions to discard the effect of random choices.

5.1. One Zone with High Demand

The simulation is performed on a network of 100 connected nodes in a ring topology. Demand decreases linearly from node 50. Two different algorithms are simulated: Weak consistency and Fast consistency.

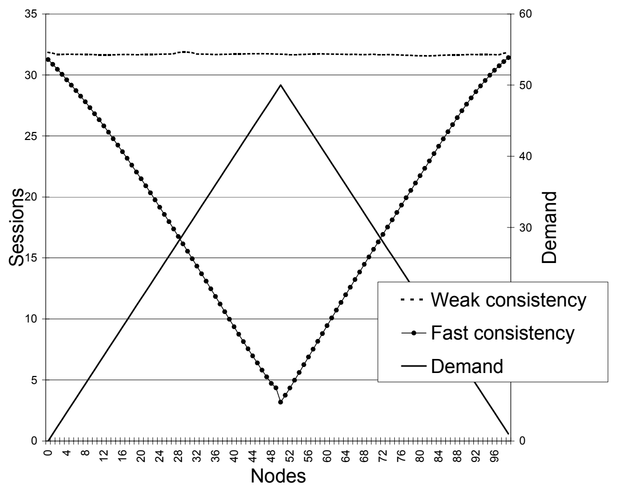

Figure 5. Number of sessions required for each node to reach a consistent state, on a ring with one zone of high demand

In figure 5 we can see that Weak Consistency is not sensitive to demand, and brings the changes to all the replicas with the same priority. Thus replicas with high demand as well as those with low demand, reach a consistent state with the same mean number of sessions. By contrast, FC which is sensitive to demand, brings updates first to replicas with high demand. Therefore, the replicas with high demand reach a consistent state with less sessions than replicas with lower demand.

5.2. Two zones with high demand

The simulation is performed on a network of 100 connected nodes in a ring topology. This time there are two zones with high demand separated by a zone of low demand. Two different algorithms are simulated: the basic Fast consistency algorithm and FC with leader interconnection.

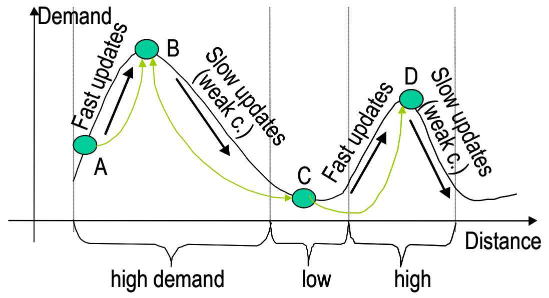

Figure 6. Shows a cross section of the distribution of demand on nodes. Two zones of high demand separated by a zone of low demand.

The expected behaviour of the basic FC algorithm is visible in figure 6: changes propagate fast (by fast update), "attracted" to higher demand zones (nodes B,D), but move as slowly as the baseline TSAE algorithm (by random anti-entropy sessions) to lower demand zones (node C).

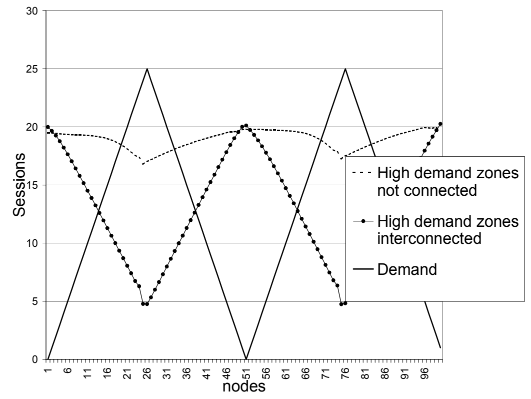

Figure 7. Number of sessions required for each node to reach a consistent state, on a ring with two zones of high demand

As we can seen in figure 7, the use of FC without interconnection of zones of high demand requires a higher number of sessions than with interconnection. This interconnection avoids that zones of low demand become barriers for FC.

Equivalent results have been obtained with a grid and linear topology with different number of nodes. When demand is the same for all the nodes, the FC algorithm has the same performance as the baseline weak consistency algorithm. But when demand changes, our algorithm systematically outperforms the baseline.

The remaining experiments are going to prove the advantages of our algorithm in more realistic conditions: a larger network (17x17 nodes) and a random demand with fractal distribution.

5.3. Grid with Fractal Demand

A fractal demand is assigned to each node. In other words, each node no longer possesses the same demand as the rest of the nodes on the grid, with several random regions of high and low demand. In this scenario, the basic “fast consistency” (FC) shows a better performance than the weak consistency algorithms (WC), but without being optimal, owing mainly to the presence of multiple high and low demand zones which cause the messages carrying the changes to move quickly towards the high demand zones, and at the speed of the WC algorithm towards the low demand zones [13].

In the FC algorithm high demand zones on the network reach a consistent state in a shorter period of time. This occurs without any increase in use of resources for carrying out this task. Thus the utility function *U(t)* grows much faster with FC.

5.4. Grid with Fractal Demand with Leaders

Since we now have the leader nodes, experiments to determine the improvement caused by the leader interconnection algorithm and the effects of the topologies of leader nodes can now be carried out.

5.5. Leader-node interconnection

Leader nodes having a number of votes greater than or equal to the average are selected and interconnected in a ring topology, which joins together the zones of high demand. In this topology, each leader node sees the same network diameter. However, there exists the disadvantage that the diameter is very large for the same number of nodes than other topologies. Simulation results can be seen in Figure 9. We can see in terms of *C(t)* or *U(t)* that FC has a better performance than WC, and also the effects of the change on the topology when the leader nodes are connected in a ring.

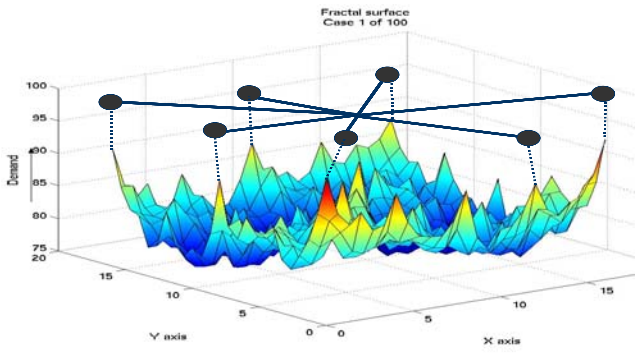

Figure 8. Fractal demand of a grid. Z-axis corresponds to the demand. The hills are high demand zones. The black dots represent leaders in logical star topology interconnection.

The same experiments have been carried out when the leader nodes are interconnected in star topology (Fig. 10).

In this topology the diameter of the leader-node network is very low, with a maximum of two. We initially though that the behaviour would be better than that obtained by the leader nodes connected in a ring topology. However, this is not the case.

In the following experiments, the leader node star topology is not included since it shows a performance inferior to that of the leader nodes connected in a ring topology.

5.6. Effect of the threshold on the choice of Leader Nodes

For this experiment, the leader nodes receiving the number of votes greater than or equal to twice the average are selected and connected together in the shape of a ring. Results are contrasted with those of the previous experiment, in which the leader nodes chosen were those corresponding to the average. With a greater threshold, fewer leader nodes are elected; they are reduced almost by half. The results can be seen in Fig. 11. They clearly show that better results are obtained with the threshold fixed at the average of votes obtained.

5.7. Second Leader Hierarchy

In the leader-node network in our fractal demand distributions, zones of high and low demand can also be distinguished. In large scale grids, it may be important to construct a second leader node hierarchy on the first, in order to connect the high demand zones of the first leader-node network. In this experiment, this second hierarchy is constructed. The results can be seen in Fig. 12. A slight improvement can be observed. Although small, this slight improvement can in fact be significant on a large scale system.

6. Related work

In OptorSim [30] a replication model for data-intensive grid applications based on economic concepts is proposed, in which an economic value is assigned to each of the files according to the number of times each file is required. Taking this value into account, we are able to decide if the file in question has to be replicated or not. Replication provides an important improvement in the time required to run data-intensive tasks. They conclude that the economic model they used has a performance at least as good as that of traditional methods. OptorSim has explored the value of predicting future demand based on historical data. This is could provide an opportunity for improvement over our simple model of demand.

In TACT [9] the aim is to adapt replication systems to clients' needs, for example, they 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 [17], clients can create replicas dynamically, whenever and wherever they are necessary. When a client sees that performance is falling, either because of increasing demand on the network, or because of client mobility, he can create replicas on a way station, 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.

"Fast consistency" can be used to complement the fluid replication system, although several aspects should be investigated: delay introduced; bandwidth consumption due to the creation of replicas by clients in our context, which is not a distributed file system. The creation of very large replicas could lead to very long anti-entropy sessions between some replicas, because of the need to transfer large databases. Even if these problems were solved, additional mechanisms would be required to coordinate replicas created by clients. These replicas will appear naturally at locations with a great concentration of clients. In such cases our algorithm could act as an excellent complement to fluid replication.

6. Conclusions

In this paper, we study the problem of propagating changes on replicated data over a network of servers using our Fast Consistency (FC) algorithm. The principal aim of FC is the propagation of changes with preference for nodes and zones of the network which have the greatest demand.

We evaluated the performance of the algorithm by simulation on several topologies (linear, ring, random, grid) using various demand workloads (constant, one zone of high demand, two zones, fractal). We have obtained the following results:

One Zone with High Demand: the basic FC algorithm outperforms the baseline for higher demand nodes.

Two zones: the leader interconnection algorithm provides an additional improvement to avoid the barrier effect of high demand zones separated by low demand zones, producing the effect of a virtually single high demand zone.

The FC algorithm has been evaluated under more realistic and extreme conditions with a large grid and fractal distribution of demand.

Grid with Flat Demand: Demand is the same for all the nodes. Here the classic weak consistency algorithm and the FC algorithm show the same performance.

Grid with Fractal Demand: The basic FC algorithm shows a better performance than the weak consistency algorithm, but without being optimal [13].

Leader-node interconnection in a Ring or Star topology: We observe an additional performance improvement, which translates in a faster growth of the utility function *U(t)*.

The results are dependent on the diameter of the network and rather independent on the number of nodes or the distribution of demand. In addition, it only requires local information, and there is no need for global coordination.

Finally we conclude that our “fast consistency” algorithm, interconnecting high demand zones by means of a logical topology, optimizes the distribution of changes by prioritizing the nodes with greatest demand. In other words, it satisfies the greatest demand in the shortest amount of time.

7. Acknowledgements

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.

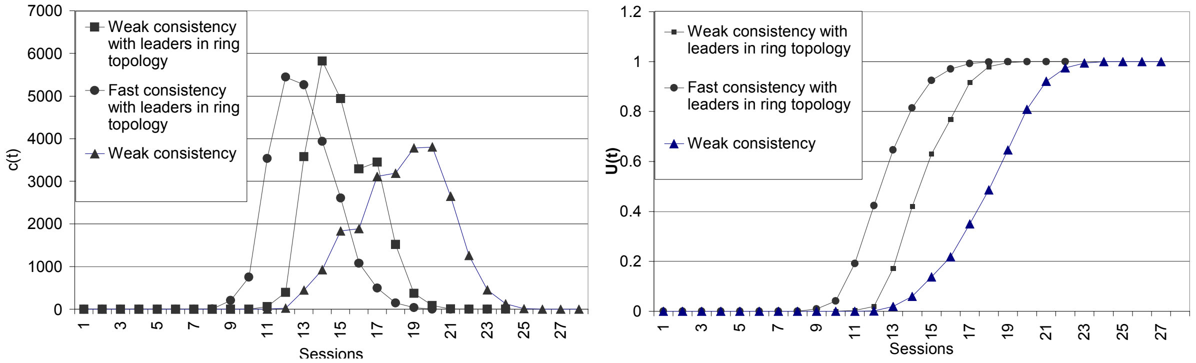

Figure 9. C(t) and U(t) to separate contributions of fast consistency and leader interconnection.

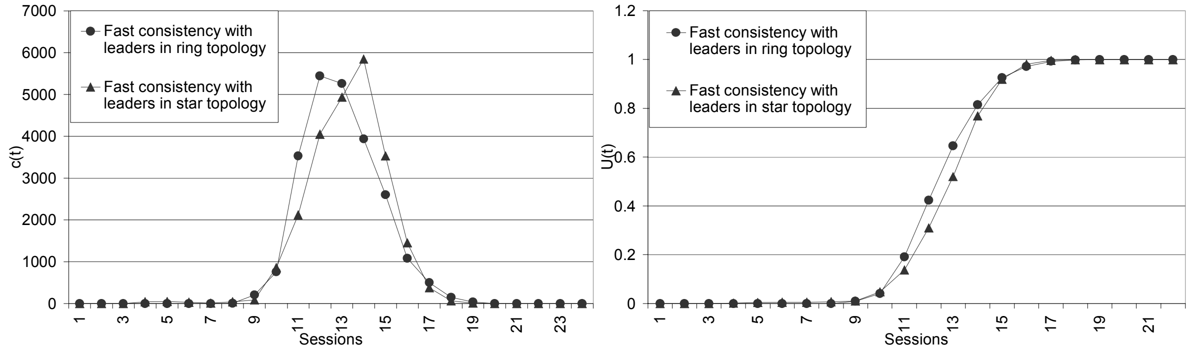

Figure 10. C(t) and U(t) for leader interconnection in ring and star topology (ring is better)

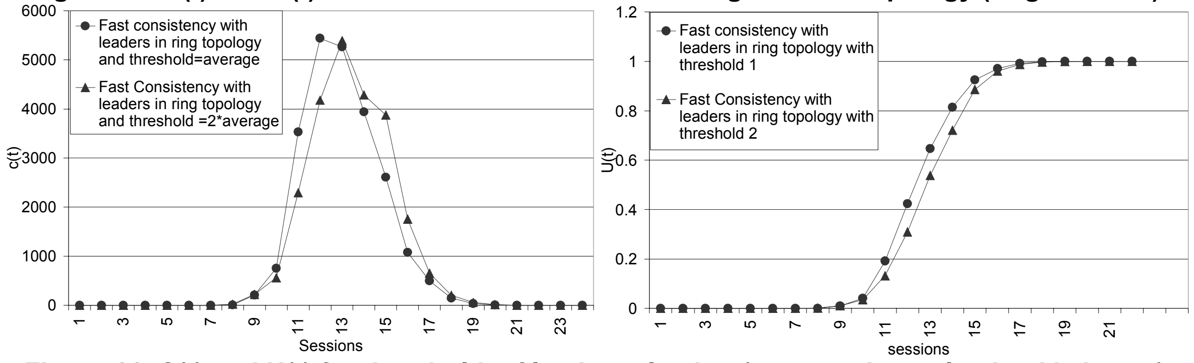

Figure 11. C(t) and U(t) for threshold of leader selection (average='more leaders' is better)

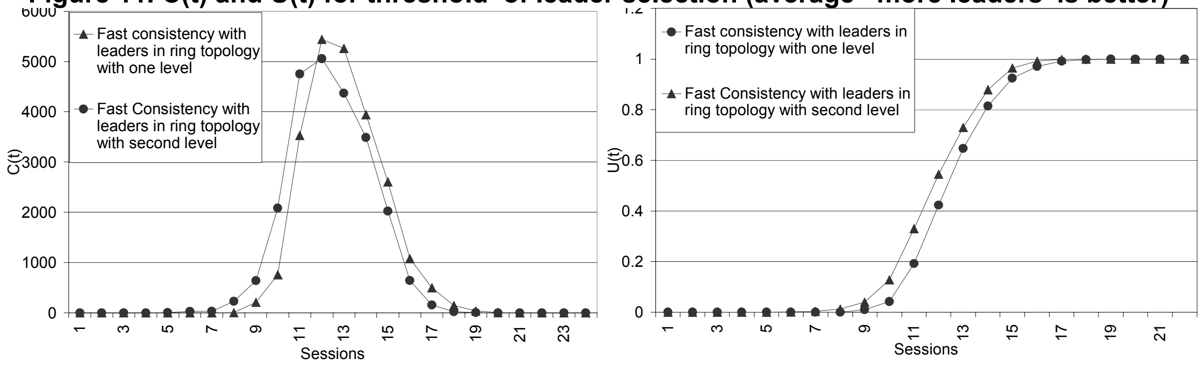

Figure 12. C(t) and U(t) for one or two levels of leader interconnection (two slightly better)

8. References

[1] Adya, “Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions”, PhD thesis M. I. T., Department of Electrical Engineering and Computer Science, March 1999.

[2] Alain Fournier, Don Fussell, and Loren Carpenter. Computer Rendering of Stochastic Models, Comm. of the ACM, Vol. 6, No. 6, June 1982, pages 371-384.

[3] Anukool Lakhina, John Byers, Mark Crovella, Ibrahim Matta.”On the Geographic Location of Internet Resources”. Internet Measurement Workshop 2002 Marseille, France, Nov. 6-8, 2002

[4] Arman Danesh, Ljiljana Trajkovic, Stuart H. Robin Michael H. Smith, Mapping the Internet, 2001.

[5] Brian Kantor, Phil Lapsley RFC0977, http://www.ietf.org/rfc/rfc0977.txt, 1986

[6] C.Neuman, “Scale in Distributed Systems. In Readings in Dist. Comp. Syst.”, IEEE Computer Society Press, 1994

[7] Ernest Chang, Rosemary Roberts, “An improved algorithm for decentralized extrema-finding in circular configurations” of proc. Comm. of the ACM May 1979 Vol. 22 Issue 5.

[8] G. LeLann, "Distributed systems - towards a formal approach," in Information Processing 77, B. Gilchrist, Ed. North-Holland, 1977.

[9] 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), Oct 2000.

[10] J. Dietterich, “DEC data distributor: for data replication and data warehousing”. In Int. Conf. On Management of data, pp 468, ACM, May 1994.

[11] Jean Laherrere, D Sornette (1998), "Stretched exponential distributions in Nature and Economy: 'Fat tails' with characteristic scales", European Physical Jour., B2:525-539.

[12] 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.

[13] Jesús Acosta Elias, Leandro Navarro Moldes, “Behaviour of the fast consistency algorithm in the set of replicas with multiple zones with high demand”, Simp. in Informatics and Telecommunications, SIT 2002

[14] Jesús Acosta Elias, Leandro Navarro Moldes, “Generalization of the fast consistency algorithm to multiple high demand zones”, ICCS2003 conference workshop "Grid Computing for Computational Science", St. Petersburg, Russia, June. 2-4, 2003.

[15] K. P. Birman, “The process group approach to reliable distributed computing”, Communications of ACM, December 1993/Vol. 36, No. 12

[16] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and Demers, “Flexible Update Propagation for Weakly Consistent Replication”, Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), Saint Malo, France, October 5-8, 1997, pages 288-301.

[17] 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

[18] M. Satyanarayanan, Scalable, Secure, and Highly Available Distributed File IEEE Computer May 1990, Vol. 23, No. 5

[19] Medina, A. Lakhina, I. Matta, and J. Byers, “BRITE: Universal Topology Generation from a User’s Pers.”.

[20] Medina, I. Matta, and J. Byers, “On the Origin of Power Laws in Internet Topologies”, ACM Computer Communication Review, pages 160-163, April 2000.

[21] 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.

[22] OpenSSL Project http://www.openssl.org/docs/crypto/RAND_bytes.html

[23] 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.

[24] 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

[25] R. Guy, J. Heidemann, W. Mak, T. Page, Jr., G. Popek and D. Rothmeier. Implementation of the Ficus Replicated File System. Proceedings Summer USENIX Conf. June 1990.

[26] Soon.-Hyung. Yook, H. Jeong, and A.-L. Barabási. Modeling the internet’s large-scale topology. Tech. Report cond-mat/0107417, Cond. Matter Archive, xxx.lanl.gov, July 2001.

[27] The Network Simulator: http://www.isi.edu/nsnam/ns/

[28] V. Duvvuri, P. Shenoy and R. Tewari, “Adaptative Leases: A Strong Consistency Mechanism for the World Wide Web”, IEEE INFOCOM 2000, pages 834-843.

[29] William H. Bell (1), David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, Floriano Zini, Simulation of Dynamic Grid Replication Strategies in OptorSim, 3rd International Workshop on Grid Computing, Baltimore, MD, 18th November 2002.

[30] Walter Willinger, Ramesh Govindan, Sugih Jamin, Vern Paxson, and Scott Shenker Scaling phenomena in the Internet: Critically examining criticality. PNAS, February 19, 2002, vol. 99, suppl. p. 1 2573–2580.