Performance Evaluation of a Hierarchical Replication Protocol: Synchronous versus Asynchronous

Noha Adly* Jean Bacon* Magdy Nagi‡

* University of Cambridge Computer Laboratory
Cambridge CB2 3QG, UK

‡ Computer Science Department Alexandria University
Alexandria, Egypt

Abstract

This paper evaluates the performance of HARP, a hierarchical replication protocol based on nodes organized into a logical hierarchy. The scheme is based on communication with nearby replicas and scales well for thousands of replicas. It proposes a new service interface that provides different levels of asynchrony, allowing strong consistency and weak consistency to be integrated into the same framework. Further, it provides the ability to offer different levels of staleness, by querying from different levels of the hierarchy. We present results from a detailed simulation analysis evaluating the benefits and losses in performance resulting from using synchronous versus asynchronous operations within HARP as well as comparing it with a traditional replication protocol.

1 Introduction

Data is replicated in distributed systems to improve system availability and performance. In recent years, the growth of internetworks and distributed applications has increased the need for large scale replicated systems. However, existing replication protocols do not address the scale and autonomy issues adequately. Communication with local replicas is usually much faster than accessing replicas on an internetwork. Therefore, protocols based on communication with *all* or distant replicas should be avoided in internetworks.

There are two approaches for managing replicated data. *Strong consistency* protocols enforce strict serializability by means of quorums [14, 8, 18] and require *synchronization* among a large number of replicas in order to ensure that replicas are mutually consistent. Synchronization across internetworks is inappropriate for large scale systems. They suffer from high latency and low throughput and a large number of replicas generate considerable traffic over the network.

In contrast, *weak consistency* protocols provide higher availability and better response time by allowing updates and queries to occur *asynchronously* at any replica. However, this approach is based on the assumption that the applications can tolerate some inconsistency and reconciliation methods should be available to resolve conflicts. Typical applications that have used weak consistency are internet news, air traffic control, airline reservation and stock exchanges. Several weak consistency protocols were presented in [10, 11, 15, 19, 22, 23]. However, they require a node to communicate with *all* nodes which will incur a large communication overhead for wide area networks like the Internet.

This paper evaluates the performance of HARP, a hierarchical replication protocol that scales well for thousands of replicas while ensuring reliable delivery. It is based on communication with nearby replicas, providing low access latencies and limiting message traffic. The protocol provides an efficient propagation scheme that is based on replicas organized into a logical hierarchy where each node has to communicate with a few nodes only (in the local proximity). Further, it proposes a novel service interface that provides *asynchronous operations* that commit at any replica then propagate to other replicas and *synchronous operations* that commit after a quorum is assembled from nodes of the top level of the hierarchy then propagate downward. This interface gives an application the ability to select strong or weak consistency semantics. Synchronous operations can scale better than traditional approaches since HARP assembles quorums from a subset of the replicas rather than from all replicas. Due to the hierarchical pattern of propagation, the scheme provides the ability to locate replicas that are more up-to-date than others, depending upon the needs of various applications.

We evaluate the benefits and losses in performance resulting from using synchronous versus asynchronous operations under different system configurations and load mixes. Further, we compare synchronous operations within HARP with the traditional majority quorum consensus scheme. Section 2 gives a brief description of the propagation scheme and interface of HARP. Section 3 describes the details of the simulation model. Section 4 presents the conducted experiments and results. Finally, section 5 concludes the paper and presents work in progress.

2 Protocol description

The scheme is based on organizing the replicas into a logical, multilevel hierarchy. In this hierarchy, nodes are grouped into clusters, and clusters are organized into a tree, such that

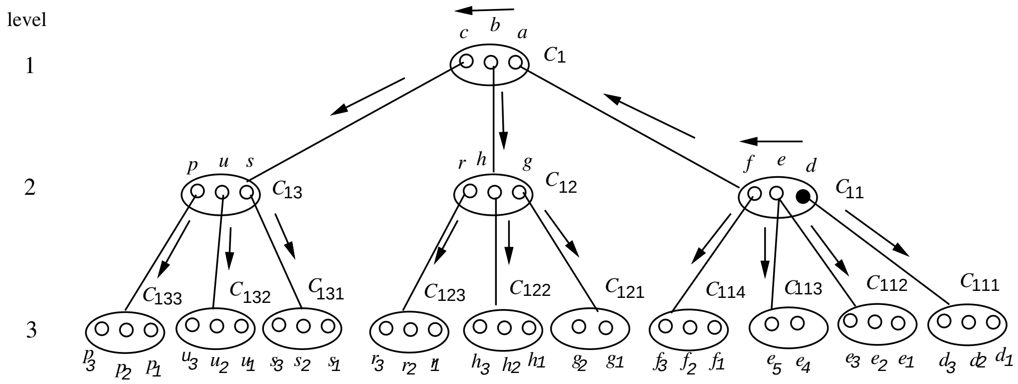


Figure 1: 40 nodes organized in a hierarchy of three levels

each cluster is assigned a father node in its parent cluster. Nodes in the same cluster should have efficient communication between them as well as with their father.

2.1 The propagation scheme

The propagation scheme is responsible for spreading updates to every node in the network while ensuring reliable delivery. The algorithm works as follows: a node i, originating a message, sends it to its neighbors, father and sons, and then waits for their acknowledgments (timeouts and retransmissions are used to overcome lost messages). Each receiving node j sends an acknowledgment to the sender, then it passes the message to the next level as follows: if the message comes from a neighbor or from the father, then j sends the message to its sons; else, if it comes from a son then j sends it to its neighbors, its father and to its sons of clusters other than the one the message comes from. This works recursively and a message originating at any site will eventually propagate everywhere. Figure 1 shows 40 nodes organized into a hierarchy of three levels with the path that a message would follow while propagating if it originated at node d. Details of the algorithm, proof of correctness as well as restructuring operations to build and reconfigure the hierarchy dynamically can be found in [2, 3].

2.2 Synchronous and asynchronous operations

Different applications require different degrees of consistency. HARP supports a set of operations that provide different levels of asynchrony allowing strong and weak consistency to be integrated into the same framework. Hence, it gives the application the flexibility to tailor the service to achieve the required degree of consistency by choosing the appropriate operations to manage its data. The following operations are supported:

- **Fast_Write**: a Fast_Write can be initiated and committed at any replica (the site of origin), and then propagated to other replicas using the propagation scheme.
- **Fast_Read**: returns the value read from any replica. It is associated with a parameter x, denoting the node to read from. The default of x is the local node.
- **Slow_Write**: a Slow_Write can be initiated by any replica; but does not commit until a quorum has been assembled from the $root$ nodes at the top of the hierarchy. For node x to perform a Slow_Write, it sends the message to its ancestor node y in the top cluster, which acts as the *coordinator* (CO) of the transaction. Node y assembles a quorum from its neighbors (using majority quorum consensus or any other standard method). Once the nodes of the top cluster commit the update, they propagate it down the tree and the initiating node commits the update on receipt of the commit message.
- **Slow_Read**: returns the value read from a quorum of nodes of the top cluster. A node x, wishing to perform a Slow_Read, sends the request to its ancestor node y in the top level. Node y assembles a quorum from its neighbors and returns the read value to x.
- **Opt_Write**: is similar to Slow_Write, but it is applied to the database of the site of origin and, optionally, to some other selected replicas (called *optimistic nodes*) before commitment. However, these updates are subject to being $undone$ if while assembling the quorum at the top level, the decision was to abort the update. Then, every replica that has *done* the update must $undo$ it.

Slow_Write should be used by operations that are likely to result in conflict and where no conflict reconciliations are acceptable. Opt_Write can be used by operations that conflict with very low probability; however, if conflicts occur their semantics can accept an abort. If an application restricts all its operations to Slow_Write and Slow_Read, then *strong consistency* (strict serializability) is achieved. It should be noticed that, by assigning the top cluster an adequate number of nodes, this scheme will result in better availability than the primary copy method and better latency and communication overhead than the usual quorum consensus methods since the number of nodes to synchronize with is small. Applications that can tolerate a lower degree of consistency can use Fast_Write, Fast_Read and Opt_Write.

Due to the hierarchical pattern of propagation that the updates follow, the scheme provides the ability to locate replicas that are more up-to-date than others while propagation is in progress. Nodes in a high level of the hierarchy usually offer better information than nodes in low levels. Therefore, Fast_Read can be used to provide different levels of staleness, depending upon the needs of various applications, by reading from replicas at different positions in the hierarchy. In this study, we evaluate two Fast_Reads, Fast_Read_0 which reads from the local node and Fast_Read_1 which reads from a node at one level up in the hierarchy, from the father node.

3 The simulation model

A simulation model was designed and implemented for performance evaluation of the proposed protocol using the simulation package SIMPACK [13].

The model consists of *num_nodes* nodes connected by a communication network. Each node has a CPU and non-volatile storage (disk) and is assumed to store a copy of the object being replicated. CPUs are modeled as single queue with single server; communication processing has higher priority than other CPU services. Each disk has its own queue, which it serves in FIFO manner.

The system is modeled as an open system where transactions arrive with an arrival rate drawn from a Poisson distribution. The parameters *cpu_req* and *io_req* specify the average amount of CPU and IO time required for transactions. They are exponentially distributed. The fraction of read transactions is determined by the parameter *Freq_R*. *Fast_R0* of the read transactions are Fast_Read_0, *Fast_R1* are Fast_Read_1 and *Slow_R* are Slow_Reads. *Fast_W* of the write transactions are Fast_Write, *Slow_W* are Slow_Write and *Opt_W* are Opt_Write. These parameters allow the examination of a wide variation of workload mixes.

3.1 Network modeling

The physical network modeled (Net 1 in Figure 2) consists of 12 nodes, three of them fully connected through long distance links to form a backbone. The remaining nodes are grouped into 3-clusters. Within a cluster, nodes are fully connected through high speed links (LAN) and experience low delay in sending messages to each other. Each cluster is connected to a node on the backbone through a long distance link. This is similar to the structure of the Internet today: regions of high connectivity with regions weakly connected through a backbone [24, 16]. Messages are reliably delivered and FIFO channels are assumed. The logical hierarchy was chosen to match the physical hierarchy: it consists of two levels, nodes on the backbone form one cluster at level 1 and each LAN cluster forms a logical cluster at level 2. The system is modeled during normal operation; that is, without failure.

Sending messages between nodes involves network protocol and operating system overhead. This overhead is modeled as *cpu_msg* msec processing at the CPU. It is assumed that there is a multicast facility rather than point-to-point communication. Also messages encounter network delays which involve transmission delays, propagation delays and media access time. Delay over a LAN is modeled as *lan_del* msec drawn from a uniform distribution and transmitters are modeled as single queue with an infinite number of servers. For WAN links this delay is modeled as *wan_del* msec, exponentially distributed, and transmitters are modeled as single server single queue.

3.2 Overhead of the algorithms

The overhead of the **propagation algorithm** is modeled as a constant overhead (*propg_ovhd_const*), plus an overhead for every logical destination it is sent to (*propg_ovhd_var*).

The generic **Two-Phase-Commit algorithm** (2PC) [5] is adopted while collecting quorums. The overhead of 2PC at the coordinator CO is modeled as a constant overhead (*pc_ovhd_const*) plus an overhead for each participant P involved in 2PC. The overhead of acquiring and releasing locks is modeled as *cpu_lock* msec processing at the CPU. The overhead of writing on logs is ignored. The simulation maintains lock tables and explicitly simulates lock contention. For Opt_Write operations, we have selected nodes that are on the route from the origin node to its CO to be the *optimistic nodes*. When CO assembles the quorum, it decides whether the update will commit or abort according to an input parameter *pr_commit*.

Deadlock avoidance is implemented by using a timeout interval based on the following heuristic [6]: $deadlock\_timeout = avg(W) + k * \sigma(W)$, where $avg(W)$ is the average waiting time of lock requests, $\sigma(W)$ is the standard deviation of waiting time of lock requests and k is a weighting factor. When deadlock is detected, each node releases locks and aborts the transaction. We use a $backoff\_period = \alpha * deadlock\_timeout$, then restart the transaction. **Majority quorum consensus** is used to collect quorums from the top cluster for synchronous operations. The *quorum collection policy* is to send the request to exactly q nodes in parallel, selected randomly, where q is the size of the quorum.

Parameter settings are a difficult issue in distributed system simulation. Our parameters for the database and transactions are comparable with previous simulation studies such as [9, 7]. The values of parameters for the 2PC protocol are

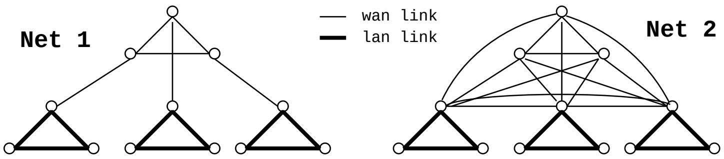


Figure 2: Physical Networks Modeled

in accordance with [9]. The communication overheads are in keeping with [17, 4, 21, 24]. The system parameters, their description and default values are listed in Table 1.

$Parameter$	Description	Default
num_nodes	Number of nodes in the network	12
inter_arr_rate	Inter arrival rate	1000 ms
Freq_R	Frequency of read transactions	.85
Fast_R0	Frequency of Fast_Read_0 trans.	var
Fast_R1	Frequency of Fast_Read_1 trans.	var
Slow_R	Frequency of Slow_Read trans.	var
Fast_W	Frequency of Fast_Write trans.	var
Slow_W	Frequency of Slow_Write trans.	var
Opt_W	Frequency of Opt_Write trans.	var
cpu_req	Processing requirements of trans	50 ms
io_req	IO requirements of trans	60 ms
cpu_msg	Overhead to send/receive a msg	2 ms
lan_del	Network delay over LAN	.02 ms
wan_del	Network delay over WAN	10 ms
prop_ovhd_const	Propagation overhead: const part	2 ms
prop_ovhd_var	Propagation overhead: var part	1 ms
pc_ovhd_const	2PC overhead: const part	2 ms
pc_ovhd_var	2PC overhead: var part	1 ms
cpu_lock	CPU time to acquire/release lock	0.2 ms
deadlock_timeout	Timeout for deadlock detection	adaptive
backoff_per	Wait period before restart	adaptive
cpu_abort	Overhead of aborting a trans.	2 ms
pr_commit	Probability Opt_Write commit	0.9
cpu_undo	CPU time to undo aborted trans.	50 ms
io_undo	IO time to undo aborted trans.	60 ms
$power\_cpu_i$	Speed of CPU at node i	1
$power\_io_i$	Speed of IO at node i	1
$power\_transc_i$	Speed of transmitter i	1
batch_length	Length of a simulation batch	2000 job
num_batch	No of batches the simulation runs	3
warmup_per	No of jobs before statistics reset	200 job

Table 1: System Parameters

3.3 Performance Metrics

The performance metrics considered are: *Response time* and *Commit time* of transactions; *Reach time*, the time at which an update reaches all replicas and $Coverage_i$ the average time at which an update originating at any node has reached i nodes. Further, we examine the probability that a Fast_Read returns out of date information as well as the *age* of the information read by Fast_Read; which is defined as the number of updates that a read has missed and is computed as: $age_j = global\_version - version_j + \gamma * abort\_age_j$. *Global_version* is a global data version counter incremented after the update commits. $Version_j$ is the current version of node j, incremented after j executes the update. It is assumed that the penalty of reading from a node that has executed x Opt_Writes which will abort, will add x units to its age multiplied by a weighting factor γ. When a node i executes an Opt_Write that will eventually abort, it increments $abort\_age_i$ which is then decremented after the node has undone the update. Also, a metric which is of interest is the *local age* which measures the number of updates missing between a node and other nodes in its neighborhood rather than the global age.

The means of the desired measurements are obtained using the methods of batch means [12]. The results reported here are within the 90% confidence interval for the quantities measured. The measurements were within 10% of their true mean in almost all cases. The simulation has been verified through an analytic model based on the *open queuing network model with multiple job classes* [20]. Details of the analytic model and results can be found in [1].

4 Experiments and results

In this section we present a selection from our performance results under several parameter variations such as read and write mixes, communication overhead as well as a comparison with the traditional majority quorum consensus protocol.

4.1 Varying read mixes with Slow_Write

This experiment explores the effect of varying *Slow_R*, *Fast_R0* and *Fast_R1* for *Slow_W*=1 and *Freq_R*=.85. We carry out three experiments:
A) *Fast_R1*=0, we vary *Fast_R0* versus *Slow_R*
B) *Fast_R0*=0, we vary *Fast_R1* versus *Slow_R*
C) we vary *Fast_R* versus *Slow_R* from 0 to 1 with fast reads split evenly between *Fast_R0* and *Fast_R1*.

Results are plotted in Figure 3. As *Fast_R* increases, the average response time of reads and writes decreases substantially. For instance, for a point where *Slow_R*=0.5, there is an improvement in read response over *Slow_R*=1 of 56% when all fast reads are Fast_Read_0 (Exp A), 41% when all fast reads are Fast_Read_1 (Exp B) and 48% when fast reads are a combination of both (Exp C). This improvement is due to the reduction of the load on the top cluster nodes and communication links as Fast_Reads are introduced. Also, the data contention decreases as *Slow_R* decreases, which improves the commit time of Slow_Read and Slow_Write. Consequently, the time at which updates reach all replicas has shown a significant improvement. The penalty paid by converting Slow_Read to Fast_Read is that fast reads return old information. It is observed that for a point of

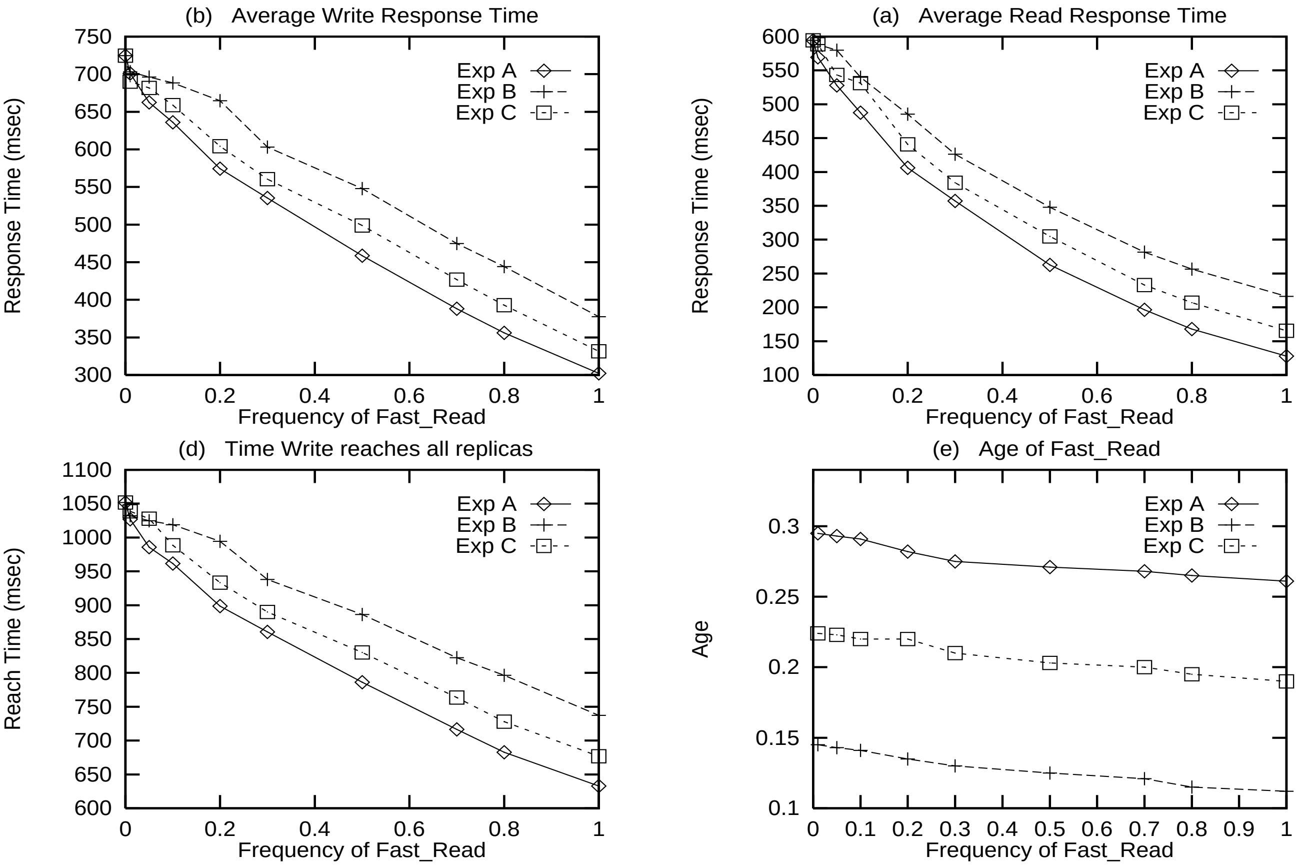


Figure 3: Varying read mix, writes are Slow_Writes

Slow_R=0.5; the probability of Fast_Read returning old information is quite low and the age of information returned by Fast_Read is around .27 (Exp A), .12 (Exp B), and .2 (Exp C) as shown in Figure 3. So, only a fraction of an update is missed (Age=0.2 could be interpreted as one update is missed once in five reads). As *Fast_R* increases, the age decreases: since the time at which updates reach all replicas decreases, replicas are more up-to-date more quickly and hence less stale.

Fast_Read_1 has better age than Fast_Read_0 while Fast_Read_0 has better performance in terms of response time. Therefore, an even mix of *Fast_R0* and *Fast_R1* appears to be a good compromise that combines the two benefits. As shown above, converting Slow_Read to Fast_Read improves all performance metrics with a little loss in age. So, the choice for the mix of *Slow_R* versus *Fast_R* depends on the application. It appears that having *Fast_R* ranging from 0.5 to 0.8 is a reasonable range that combines both benefits.

4.2 Varying *Opt_W* versus *Slow_W*

In this experiment, *Slow_W* versus *Opt_W* is varied to study the performance of a combination of load mix between them. Parameters are set for *Freq_R*=.85, *Slow_R*=0.5, *Fast_R0*=.25, *Fast_R1*=.25 and *pr_commit=0.9*.

The average Write response time decreases linearly as *Opt_W* increases. Further, the coverage time decreases, especially for the first few nodes. However, the average age increases as *Opt_W* increases. The reason is that updates commit at local replicas and update the global version rather than updating it while assembling the quorum. The local age is slightly better with Opt_Write due to the improvement in the coverage time. From this experiment it was concluded that the ratio of *Slow_W*:*Opt_W* ranging between 4:6 up to 6:4 is a good mix.

4.3 Varying the network delay

In this experiment, we explore the sensitivity of performance to the delay on the network. We fix the *lan_del* to 0.02 msec and vary the *wan_del* from 0.2 to 100 msec. Parameters are set to *Opt_W*=*Slow_W*=0.5 and *Freq_R*=.85. Results are shown for two load mixes:

A) *Slow_R* =.2, *Fast_R0*=.4, *Fast_R1*=.4

B) *Slow_R* =.5, *Fast_R0*=.25, *Fast_R1*=.25

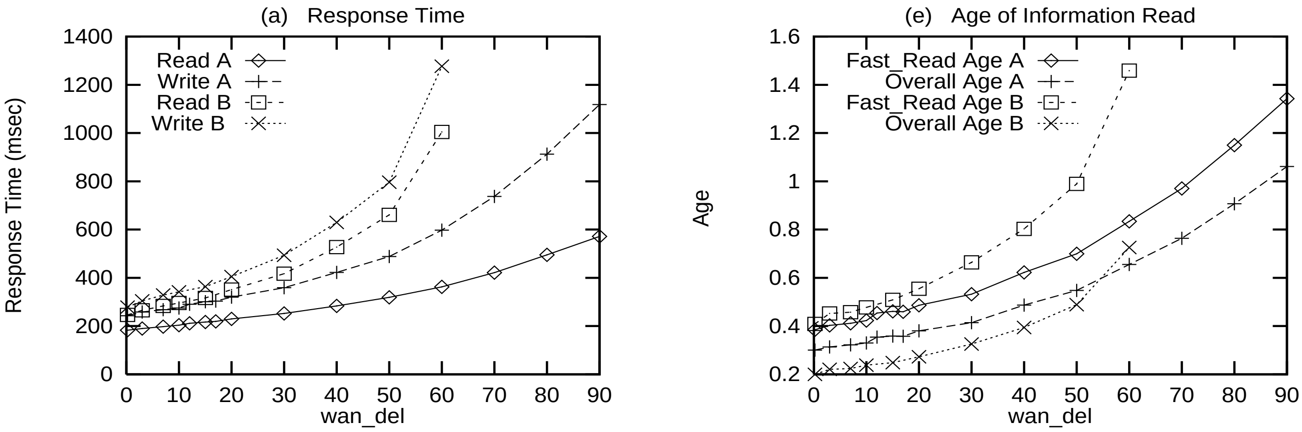


Figure 4: Varying the network delay

As shown in Figure 4, the response times of reads and writes are very sensitive to *wan_del*. They are quite low for values of *wan_del* less than 20 msec then, they increase sharply. It is obvious that load B is more sensitive to *wan_del* than load A and saturation is reached more quickly. It is observed that the responses of Fast_Read_0 and Fast_Read_1 increase only slightly, while those of Slow_Write and Slow_Read increase sharply and dominate the results. The resources were lightly loaded, even underutilized, with data contention as the bottleneck. The reason is that: while *wan_del* increases, the time it takes a transaction to assemble a quorum and commit increases and it holds locks for a longer time. Consequently, the time other transactions wait for locks increases. The age returned by Fast_Read with B is higher than A since updates take longer to commit and consequently take longer to reach other replicas. The overall age of B is slightly lower than A (0.1 updates) for low values of *wan_del* then the gap decreases and eventually it becomes higher than A for high values of *wan_del*.

From here we can deduce that performance is very sensitive to *wan_del*. Synchronous operations slow down the system operation and affect the performance tremendously. Response time deteriorates as well as age while resources are underutilized and throughput is limited due to data contention. This suggests that if applications can accept observing slightly stale information then they should increase their frequency of Fast_Read especially when the network delay becomes higher as this offers better performance as well as lower age. Similar conclusions were obtained by varying *cpu_msg*.

4.4 Varying network topology

This experiment shows the performance over a network where all nodes are in one logical cluster, more or less fully connected as shown in Figure 2 (Net 2). Hence, applying slow operations on such a network represents the performance of standard protocols used in the literature, more precisely, using majority quorum consensus over the whole network. We compare results with those of Net 1. Parameters are set to *Freq_R*=.85, *Slow_W*=*Opt_W*=.5 and we vary *Slow_R*, *Fast_R0* and *Fast_R1* such that *Fast_R0* and *Fast_R1* are split evenly. The results showed that:

* Steady state was not able to be reached on Net 2 for *Slow_R* higher than 0.5. Assembling a quorum from 7 nodes (versus 2 in Net 1), increases data contention. Further, since each node can be a CO, the rate at which deadlock occurs is higher. Finally, resources are more loaded since all nodes are involved with 2PC and communication overhead. These factors increase the time by which transactions commit and eventually no steady state could be reached.

* For loads where steady state could have been reached, most of the performance metrics showed a degradation. The average response time of Writes is far higher than in Net 1: nearly double as shown in Figure 5 due to high commit time. Although Net 2 needs to cross fewer links than Net 1 to propagate updates to other replicas not participating in the quorum; the increase in commit time is higher than the reduction in the communication time. The age returned by Fast_Read in Net 2 is lower than that of Net 1 (around 0.2 updates for low *Slow_R* and 0.17 updates for high frequency). Since a majority of the nodes participate in the quorum, they are up-to-date, and since Fast_Read is performed directly from them, this explains the lower age.

In conclusion, it is observed that performance over Net 2 is much worse than Net 1. Throughput is limited tremendously, especially with high loads. The performance of Slow_Write, Slow_Read, Fast_Read_0 were worse than Net 1, as well as the time at which updates reached all replicas. The decrease in the response time of Fast_Read and the gain in the age over Net 1 is considered to be much less than the losses in all other metrics.

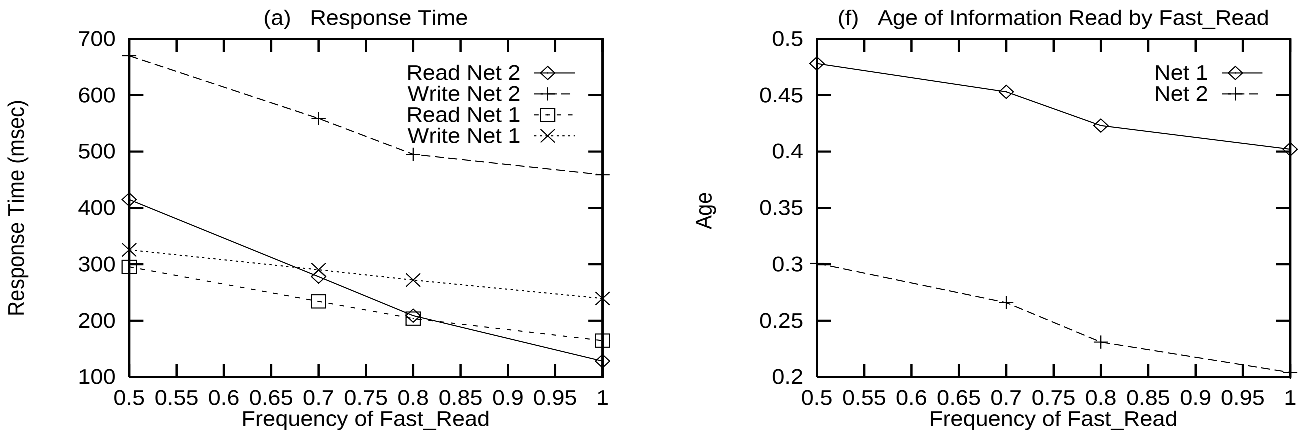


Figure 5: Varying network topology

Several other experiments were done with different network topologies, details of which can be found in [1].

4.5 Comparing Slow_Write versus Fast_Write with a mix of Fast_Reads

In this experiment, we vary a mix of *Fast_R0* and *Fast_R1* associated with A) *Fast_W*=1 and B) *Slow_W*=1 for comparison. *Freq_R* is set to 0.7, *Slow_R* to 0 and we vary *Fast_R0* versus *Fast_R1*.

There is a huge difference between write response of Fast_Write and Slow_Write as shown in Figure 6. The average response time of fast reads with Slow_Write is slightly higher than with Fast_Write because resources with slow writes are more loaded due to 2PC and communication overhead. The time at which updates reach all replicas is much higher for Slow_Write than Fast_Write and it increases sharply as *Fast_R1* increases. This is mainly due to the load slow writes put on nodes of the top cluster which increases commit time and therefore reach time. The age returned by Fast_Reads with Slow_Write have better age than those with Fast_Write. This is expected as replicas in the top cluster with Slow_Write are updated first, hence age (especially that of Fast_Read_1) is better. However, since the reach time is better than Slow_Write, replicas with Fast_Write catch up quickly. It is observed however, that the age is only a fraction of an update and that the difference between the age of reads between the two writes ranges from 0.25 to 0.5 updates.

In conclusion, with Fast_Write every performance metric is better than with Slow_Write except the age. However, the loss in age is low and the benefits are much higher than the losses. We reached the same conclusion regarding Fast_Read_0 and Fast_Read_1: Fast_Read_1 offers better age with a slight loss in the response time.

Several other experiments were conducted, see [1], and it was found that Fast_Write is less sensitive to system parameters than Slow_Write and can achieve better throughput and performance. Also, the performance of Fast_Write on several network topologies was evaluated. A network of 39 nodes with a logical hierarchy of three levels was also used and similar conclusions were obtained.

5 Conclusion and Work in Progress

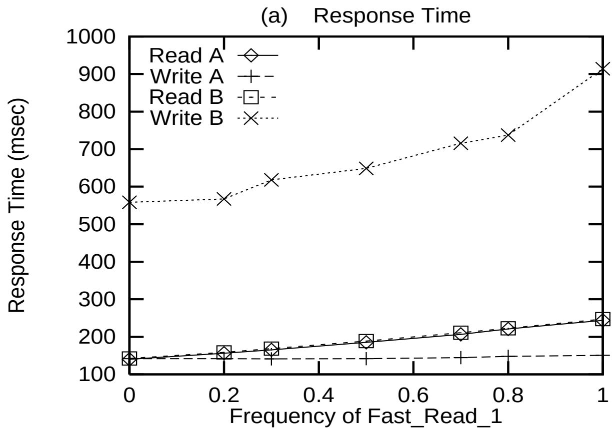
A hierarchical protocol for managing large scale replicated data was presented where each node has to communicate with only a few other nodes, typically ones in its physical proximity, thus reducing communications overhead and improving latency. A new service interface is proposed that allows the application to select its required degree of consistency and to offer different levels of staleness by querying from different levels of the hierarchy.

A detailed simulation study was performed to evaluate the benefits and losses resulting from using synchronous versus asynchronous operations under different system configurations and load mixes. Asynchronous operations have shown much better performance than synchronous and can cope with heavily loaded systems with a little loss in reading slightly out of date information. Using the principle of locality, HARP provides read operations that return less stale information with little increase in response time. Finally, synchronous operations within HARP perform better than traditional protocols since it limits quorum assembly to a small number of nodes.

Currently, we are evaluating the performance of a weak consistency protocol to compare it with HARP. The *Time Stamped Anti Entropy* (TSAE) [15] protocol has been chosen for this purpose and analysis is still underway.

Acknowledgments

Noha Adly has been supported by an FCO grant from the British Council.



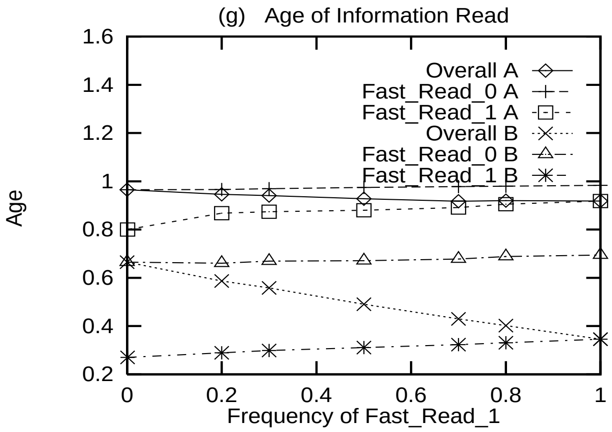


Figure 6: Comparing Slow_Write versus Fast_Write with a mix of Fast_Reads

References

[1] N. Adly. *Management of replicated data in large scale systems*. PhD thesis, Computer Laboratory, Cambridge University. under preparation.

[2] N. Adly. HARP: a hierarchical asynchronous replication protocol for massively replicated data. Technical Report TR-310, Computer Laboratory, University of Cambridge, UK, August 1993.

[3] N. Adly, M. Nagi, and J. Bacon. A hierarchical asynchronous replication protocol for large scale systems. In *Proceedings of the IEEE Workshop on Parallel and Distributed Systems*, pages 152–157, Princeton, NJ, October 1993.

[4] A. Agrawala and D. Sanghi. Networks dynamics: an experimental study of the internet. In *IEEE Proceedings of GLOBECOM'92*, pages 782–786, December 1992.

[5] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley, 1987.

[6] M. Carey, B. Lindsay, and R. Obermarck. The performance of alternative strategies for dealing with deadlocks in database management systems. *IEEE Transactions on Software Engineering*, 13(12):1348–1363, December 1987.

[7] M. Carey and M. Livny. Conflict detection tradeoffs for replicate data. *ACM Transactions on Database Systems*, 16(4):703–746, December 1991.

[8] S. Cheung, M. Ammar, and M. Ahamad. The Grid protocol: a high performance scheme for maintaining replicated data. In *Proceedings of the IEEE 6th Int. Conf. on Data Engineering*, pages 438–445, 1990.

[9] B. Ciciani, D. Dias, and P. Yu. Analysis of replication in distributed database systems. *IEEE Transactions on Knowledge and Data Engineering*, 2(2), June 1990.

[10] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Symposium on Principles of Distributed Computing, ACM SIGACT-SIGOPS*, pages 1–12, 1987.

[11] A. Downing, I. Greenberg, and J. Peha. OSCAR: an Architecture for Weak Consistency Replication. In *Proceedings of IEEE PARABASE-90*, pages 350–358, March 1990.

[12] D. Ferrari. *Computer systems Performance Evaluation*. Prentice Hall, 1978.

[13] P. Fishwick. Simpack: getting started with simulation programming in C and C++. In *Winter Simulation Conference*, pages 154–162, Arlington, VA, December 1992.

[14] D. K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the 7th Symposium on Operating System Principles*, December 1979.

[15] R. Golding. *Weak-consistency group communication and membership*. PhD thesis, University of California, Santa Cruz, 1992.

[16] R. Golding and D. Long. Accessing replicated data in large scale distributed system. *International Journal in Computer Simulation*, 1(2), June 1991.

[17] J. Gray. The cost of messages. In *ACM Symposium on Principles of Distributed Computing*, pages 1–7, August 1988.

[18] A. Kumar. Hierarchical quorum consensus: a new algorithm for managing replicated data. *IEEE transactions on Computers*, 40(9):996–1004, September 1991.

[19] R. Ladin, B. Liskov, and L. Shrira. Lazy replication: exploiting the semantics of distributed services. In *Proceedings of the 9th ACM symposium on Principles of Distributed Computing*, pages 43–57, Quebec City, CA, August 1990.

[20] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, 1984.

[21] C. Pu, F. Korz, and R. Lehman. a measurement methodology for wide area networks. Technical Report CUCS-044-90, Columbia University, New York, NY 10027, March 1991.

[22] P. Queinnec and G. Padiou. Flight plan management in a distributed air traffic control system. In *Proceedings of the International Symposium on Autonomous Decentralized Systems*, Kawasaki, Japan, March 1993.

[23] G. Wuu and A. Bernstein. Efficient Solutions to the Replicated Log and Dictionary Problems. In *Proceedings of the third ACM Symposium on Principles of Distributed Computing*, pages 233–242, August 1984.

[24] Yongguang Zhang. *Communication Experiments for Distributed Transaction Processing – from LAN to WAN*. PhD thesis, Purdue University, December 1994.