Consistent Data Replication: Is it feasible in WANs?

Yi Lin[1], Bettina Kemme[1], Marta Patiño-Martínez[2], and Ricardo Jiménez-Peris[2]

[1] McGill University, School of Computer Science, Montreal, Quebec, Canada*
[2] Facultad de Informatica. Universidad Politecnica de Madrid , Spain**

Abstract. Recent proposals have shown that database replication providing 1-copy-serializability can have excellent performance in LAN environments by using powerful multicast primitives. In this paper, we evaluate whether a similar approach is feasible in WAN environments. We identify the most crucial bottlenecks of the existing protocols, and propose optimizations that alleviate the identified problems. Our experiments show that performance remains acceptable even for medium sized systems, and data replication guaranteeing 1-copy-serializability is a serious alternative to weaker approaches in WAN environments.

1 Introduction and Motivation

With the wide use of online transaction processing systems (e.g., online stores), comes the need for fault-tolerance, scalability, and fast response times. Replication is the most common approach to achieve these properties. For instance, if a retailer replicates its data at the different store locations, each store has fast local access, and the system can survive site crashes. For such applications, data consistency despite high update loads, and the flexibility to submit any transaction to any database replica are more important than unlimited scalability. That is, having two to ten replicas is probably a typical system configuration.

A big challenge is to keep the data replicas consistent. There is a trade-off to pay between providing full data consistency (e.g., 1-copy-serializability and atomicity) and fast response times [13]. In recent years, many replication protocols have emerged [2, 4, 17, 11, 14, 6, 16, 21, 10, 3, 23, 22] providing both data consistency and excellent performance in LANs. Many of these protocols determine the serialization order at the begin of transaction, and then serialize transactions according to this order. This is faster than traditional distributed locking and avoids an expensive 2-phase commit protocol.

However, little research has been done whether these solutions can also be applied to WANs. [1] analyzes one particular protocol. [23] uses multicast protocols that have been developed for WANs. Other solutions usually execute and commit transactions at one site, and propagate changes to other sites only some time after commit [24] allowing for faster response times. This is also referred to as lazy replication. However, remote sites might have stale data, and committed transactions might be lost in case of crashes. Lazy approaches also often restrict updates to be executed at a single primary

* Research partially funded by MDER-Programme PSIIRI: Projet PSIIRI-016/ADAPT, by NSERC Rgpin 23910601, and by FQRNT 2003-NC-80398.

** Research partially funded by the European Commision under project Adapt IST-37126 and by the Spanish Research Council (MEC) under project TIN- 2004-07474-C02-01.

site. This, however requires a client to send its update transactions over the WAN if it is not located close to the primary, again increasing response times. Other approaches allow inconsistencies between replicas [12, 24] which are difficult to resolve.

Considering these shortcomings of existing WAN solutions, this paper revisits the successful solutions developed for LANs, and evaluates how they perform in WANs. We have detected several shortcomings of these protocols when executed in WANs, and suggest improvements that help to alleviate the major bottlenecks. They show how the message overhead within the response time can be reduced through various means. Our performance results show that data consistency can be obtained with acceptable performance. In many of our experiments, response times remain below 1 second, and throughput increases over a centralized system. Furthermore, query (read-only) response times are excellent and not affected by update transactions.

2 Database Replication Strategies developed for LANs

One common approach to provide 1-copy-serializability is to submit all SQL requests to a central middleware [3, 6] which performs concurrency control (usually on a table basis), and forwards reads to one, and updates to all replicas. This leads to communication between middleware and database replicas for each operation within a transaction.

Alternative approaches assume a replicated middleware, where a middleware instance is installed in front of each database replica. Clients contact the closest middleware instance. A client request triggers the execution of a transaction which might contain several database statements. The transaction programs either reside within the middleware or can be called from the middleware. This is an advantage for WAN replication, since client/middleware and middleware/database communication is always local – only middleware/middleware communication is across the WAN.

Most proposals following this decentralized approach ([2, 1, 11, 14, 16, 21, 10, 23]) use group communication systems [8] (GCS). All middleware replicas build a group and multicast messages which are received by all members (including the sender). Different multicast primitives provide different *ordering* and *delivery* semantics. The ordering semantics of interest for this paper are *unordered*, and *total order* (for each two members receiving m and m', both receive them in the same order). The delivery semantics are *reliable* (whenever a member receives a message m and does not fail, then all other group members will receive m unless they fail), and *uniform reliable* (whenever a member p receives a message, all other members will receive the message unless they fail –even if p fails shortly after message reception). Uniform reliable delivery provides all-or-nothing even in failure cases, while reliable delivery allows failed members to have received messages that are not received by others. Some systems (e.g., Spread [26]), call a combination of reliable and total order *agreed* delivery, and a combination of uniform reliable and total order *safe* delivery. We adopt this notation. Note that the GCS probably needs to send more than one physical message per multicast message submitted by the application, e.g., in order to determine the total order. In general, the higher the degree of ordering and/or reliability, the more internal messages will be necessary and the higher the message delay for a multicast message. In the following analysis, we only consider the number of multicast messages submitted by the database system

and not the actual messages sent by the GCS since the latter depends on the particular algorithms implemented within the GCS.

We will now present a simple replication protocol using group communication. Most existing protocols are extensions of this basic protocol. The middleware distinguishes between read-only transactions (also called queries) and update transactions, e.g., by analyzing the SQL statements. It handles update transactions as follows.

I. *Upon receiving a request for the execution of an update transaction from the client*: multicast the request to all sites with safe delivery.
II. *Upon receiving an update request in safe delivery*: add the request to a FIFO queue.
III. *Once a request is the first in the queue*: submit the transaction for execution.
IV. *Upon finishing execution*: remove the request from the queue, and respond to client.

This protocol executes update transactions serially according to the total order multicast, and guarantees atomicity by relying on uniform reliable delivery. In order to allow transactions to execute concurrently, many approaches do concurrency control at the middleware. They often assume that the objects to be accessed are known in advance (e.g. by parsing the SQL statements to determine the tables to be accessed). The middleware can then, e.g., atomically request locks for all tables the transaction is going to access upon safe delivery. When all locks are granted, the transaction can start executing. In this case, non-conflicting transactions can execute concurrently. [14, 16, 10] follow this or similar approaches. We call this the *symmetric approach*. An example is shown in Fig. 1(a). Update transactions T1 and T2 are submitted concurrently to sites A and B respectively, which then multicast the request in total order. Both sites execute T1 and T2 according to the total order if they conflict otherwise in any order.

Queries only need to be executed at one replica. In a WAN this is probably the local replica to avoid messages. Since many database systems (e.g., Oracle and PostgreSQL) provide a special snapshot mode, in which queries read from a committed snapshot of the data, the middleware can immediately submit queries to the database replica without any further actions, and still provide 1-copy-serializabilty.

3 Replication Strategies in WANs

There exist many optimizations over the protocol above. They have been either analyzed only for LANs, or not at all. In this section, we look at several optimizations and their potential effect in WANs. We only look at update transactions since queries are local.

3.1 Communication Choices

Agreed vs. safe delivery. Safe delivery leads to long delays since it requires acknowledgements (acks) from all sites before messages can be delivered. Therefore, it is important to understand in which case agreed delivery violates transaction atomicity: a site must receive a client request, multicast it, receive it in agreed order, execute and commit the transaction, return the ok to the client and then fail while none of the surviving sites receives the request and hence, commits the transaction. Only in this case a committed transaction is "lost". This might not happen often, even in a WAN. It is up to the application to decide whether it can accept such cases or not.

Optimisitic Delivery. The idea is to deliver a message once optimistically (e.g., when it is physically received) and once final (e.g., when safe) [18]. The transaction starts executing upon the optimistic delivery but may only commit at the final delivery. This allows overlapping transaction execution and message delay. If the optimistic and final delivery orders are not the same, some transactions might have to abort. [25] delays optimistic delivery until chances are high that it has the same order as the final delivery. [19] considers uniform reliable delivery. Since out-of-order delivery is likely in a WAN, we look at a variation where the optimistic delivery is agreed (i.e., total order is established), and the final delivery is safe (i.e., uniform reliability is established).

Early Execution Variation. However, existing middleware based replication approaches based on optimistic delivery [20] do not always overlap transaction execution with the delay for final delivery. As an example, in Fig. 2(a) transaction T1 is delivered optimistically. It is executed but has to wait to commit until final delivery. T2 is delivered optimistically before T1's final delivery. If it conflicts with T1 the middleware will not start T2's execution until T1 has committed despite its optimistic delivery. Conflicts are likely if the middleware uses table level locking. However, on a tuple-basis, T2 might not conflict with T1. In LANs, this unnecessary waiting might not have a big impact because the delay between T1's optimistic and final delivery, and hence, T2's commit, is small. But in a WAN, we should take full advantage of any possible concurrency. Hence, it would be desirable to execute T2 as soon as possible We suggest as an optimization to take advantage of the concurrency control of the underlying database system if it uses tuple-based strict 2-phase locking. For example, in Figure 2(b), T2 can start execution once T1 has finished its execution. At this time, T1 has already acquired all necessary tuple locks at the database. If T2 does not conflict with T1 on any tuple, T2 can acquire all its tuple locks and execute while T1 is waiting for the final delivery of its message. If T2 and T1 conflict, T2 will block on some tuple until T1 commits and releases its locks which is the correct behavior. Hence, we will execute T1 and T2 in the correct serialization order according to the total order.

Total Order. There exist many algorithms to determine a total order (for a survey see [9, 5]). However, little has been done to evaluate them in combination with an application or in WANs. We analyze some well known algorithms. In *token* a token circulates among the sites, and only the token holder may multicast messages. Increasing sequence numbers indicate the order of these messages. In *sequencer* a sender requests a sequence number from a sequencer and then multicasts the message with this number. The *Lamport* algorithm delivers a message m at a site once this site has received messages from all other sites piggybacking acks for m. Two concurrent messages (from different senders and neither contains an ack for the other message) are delivered in predefined order, e.g., site priority. Finally, a special case of the ATOP approach [7] delivers in a *round-robin* fashion one message from each site.

3.2 Write Set Options

Alternatively to executing the entire transaction at all replicas as in the symmetric approach, execution can take place only at one replica. The resulting changes are propagated in form of a write set (i.e., set of update tuples) at the end of execution to the other

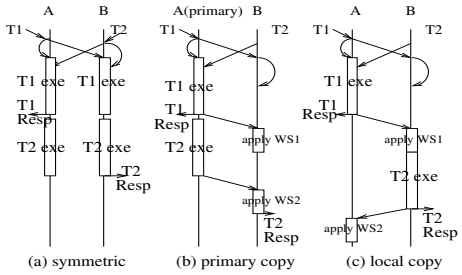

Fig. 1. With or without write set

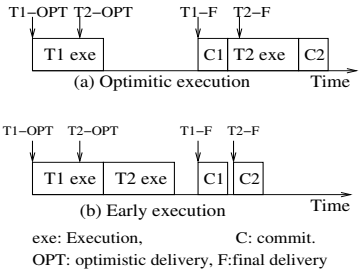

Fig. 2. Optimitic v.s. early execution

sites which apply the write set. Applying writesets is usually faster than executing all read and update SQL statements [16]. Additionally, write sets might be the only feasible solution if there exists non-determinism (e.g., set an attribute to the current local time) which leads to data divergence if SQL statements are executed at all replicas.

Primary Copy Approach. [16] uses write sets with a primary copy approach. Each table has a primary copy, and a transaction updating this table must execute on the site holding the primary copy. If a transaction T wants to access tables that have different primaries, it is executed at the primary of one of these tables. We refer to this as the primary site of T. As before, a request for transaction T is multicast in total order to all sites and the locks requested according to its total order delivery. But only the primary site executes T and multicasts the write set using unordered reliable delivery (if the site crashes, another site can simply reexecute). The primary can commit immediately. The other sites apply the write set once the locks are granted locally. Conflicting transactions might have different primary sites, but all sites will execute or apply the write sets according to the total order since all request locks in this order. Hence, serializability is guaranteed. This approach sends two multicast messages (1 total order and 1 unordered reliable) per transaction. Fig. 1(b) depicts an example. T1 and T2 are submitted to A and B, respectively, and multicast in total order. A is primary site for both transactions. If they conflict, A executes them serially, otherwise concurrently. At the end of the execution of a transaction, A multicasts the write set to the other sites in unordered reliable order. Since T1 was submitted to A, A also returns a confirmation to the client. B, upon receiving T1 and T2, requests the locks in the correct order but does not execute the transactions. Instead it waits for the write sets from A and then applies them (serially if they conflict). It also returns the confirmation for T2 to the client. For T1 the primary is the local site, hence the response time only includes the delay of the request message. For T2 the response time also includes the delay of the write set.

Adjusting to WANs. Since two messages within transaction boundaries can increase response times significantly, we propose an optimization which provides more "locality". In the *local copy approach* depicted in Fig. 1(c), a request for an update transaction is still multicast in total order, but the transaction is executed at the local site it is submitted to. In the figure, T1 executes at A, and T2 at B. Since T1's request is received before T2's request according to the total order, A first executes T1, then multicasts the write set using unordered reliable delivery, commits and returns the confirmation to the

user. If T1 and T2 conflict, B waits to receive and apply T1's write set, then executes T2, multicasts its write set and returns the confirmation to the user. There are still two multicast messages per transaction as in the primary copy approach. If two conflicting transactions are submitted concurrently at different sites as in the figure, the first one to be delivered (T1) has only the request message within the respone time, the second one must also wait for the write set of the first to arrive before it can execute locally. This overhead is similar to the primary copy approach. However, if transactions do not conflict, only the request message is delivered within the response time.

4 Experimental Results

4.1 System Description

We have developed a modular Java based middleware that allows for easy plug-in of different replication strategies. Our middleware supports the symmetric (Sym), the primary copy (PC), the local copy (LC) approach, and the early execution variation of the symmetric approach (ESym). The middleware uses table based locking. We use PostgreSQL 7.2 as database backend. We extended PostgreSQL to provide a function to get the changes performed by a transaction (write set), and a second one that takes these changes and applies them. We used two open-source group communication systems: Spread and JavaGroups. Spread [26] uses a token protocol providing agreed and safe delivery. We integrated an optimistic delivery into Spread that basically delivers a message optimistically upon agreed delivery, and finally at safe delivery. We refer to this as optsafe delivery. JavaGroups [15] implements token based and sequencer based agreed delivery. We have further implemented the Lamport (safe delivery) and the round-robin variation of ATOP (agreed delivery) on top of JavaGroups. The last two algorithms guarantee that all sites send messages regularly by sending null messages if necessary.

4.2 Experiment Setup

The experiments were conducted across the Internet on eight machines with similiar strength (e.g. AMD 1666MHZ / 512MB memory / Red Hat Linux or Solaris), each located in a different city (four in Canada, one in Spain, one in Switzerland and two in Italy). The round trip times between machines in the same continent and between different continents are around 40 ms and 150 ms respectively.

Since all standard benchmarks have at least 50% read workload, we use our own synthetic application to stress-test the system. Our database consists of 10 tables with 10000 tuples each and most experiments use only update transactions to understand the limitations of our approach. Each update transaction modifies 10 randomly selected tuples in a randomly selected table. Since the middleware uses table-based locking, the chance that two concurrent update transactions conflict is 10%. Our scalability analysis also uses queries each of which scans a whole table performing some aggregation (*select avg(attr3), sum(attr3) from* $table_i$). Execution time of this query is three times longer than for an update transaction (as typically encountered in real applications).

In each test run, clients are equally distributed among all replicas and submit transactions concurrently and with the same rate to achieve the desired system-wide load.

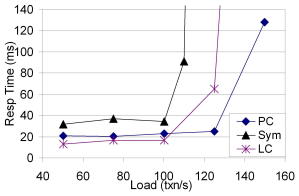

Fig. 3. Write Sets in LAN

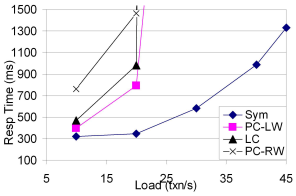

Fig. 4. Write Sets in WAN

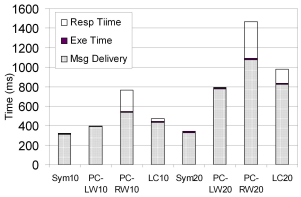

Fig. 5. Response Times WAN

Within each single machine, results were with a 95% confidence within +/- 2.5% of the results taken for our figures. However, response times for different machines varied depending on their setups, and we show the average over all machines. Most figures show the response times up to 1.5 seconds with increasing load submitted to the system. We set 1.5 seconds as an upper limit of what we consider acceptable. Often, the load could be further increased without saturation.

For baseline comparison, we conducted experiments with LAN and WAN clients that directly connect to a single database. With 100% updates, response times of LAN clients never exceed 20 ms up to the saturation point of 120 transactions per second (tps). For WAN clients, response time was over 1.5 seconds at a maximum throughput of 5 tps. This is probably due to message overhead and connection handling.

4.3 Write Set Options

We evaluated the PC, LC and Sym write set options in a LAN and a WAN with 100% updates and 5 sites. We use Spread, and safe delivery for transaction request messages. Message delivery takes a few ms in LAN but up to hundreds of ms in WAN.

Fig. 3 presents the results for the LAN. PC and LC have lower response times and achieve higher throughput than Sym. This is due to the fact that only one site executes the transactions while the others only apply the changes which requires less CPU resources. In a LAN the GCS can easily handle the high message load of LC and PC without becoming a bottleneck. LC has slightly better response time than PC at lower loads since at low loads there are few concurrent transactions, hence in LC nearly all transactions have only one message delay within their response times while for PC transactions with a remote primary always have two message delays within their response time. However, PC outperforms LC at high loads, since more concurrent transactions lead to more conflicts and hence, longer blocking times for LC.

Fig. 4 shows the results for the WAN. For PC, we have one graph showing the response time for a transaction T that was submitted to the primary of T (*local writes LW*), hence only the delay of the request message is within the response time, and one graph for a transaction T submitted by a client that is not connected to the primary of T (*remote writes RW*), hence both the delay of request and writeset message are within the response time. Contrary to the LAN environment, Sym has much better response time than the approaches using write sets. Additionally, Sym can handle up to 45 tps while PC and LC can only handle up to 20 tps with acceptable response times. PC-LW has lower response times than LC, PC-RW has the worst response time.

The reasons for this behavior can be explained by Fig. 5 which shows a detailed response time analysis at loads of 10 and 20 tps in a WAN. The response time is divided into the time needed to deliver the transaction request message (grey), the transaction execution time within PostgreSQL (black), and the remaining time (white). The latter includes the time for transactions waiting in the middleware for their turn to execute and commit. This includes, e.g., the time for a lock to be granted, or in the PC and LC approaches the time the transaction might need to wait for a write set to be delivered. The figure shows that transaction execution (black) only takes a small percentage of total response time for all approaches. That is, the PostgreSQL databases were never the bottleneck. In all cases the time for safe delivery of the request message (grey) has the biggest impact on the response time. Furthermore, PC and LC have generally longer delays for the safe delivery and resulting longer response times than Sym because Spread had to handle double as many multicast messages than with Sym, and hence, was higher loaded. This became especially significant at 20 tps. Using Sym, barely any time is spent in the middleware (white) waiting for locks or similar. In regard to PC, PC-LW transactions do not have any delays in the middleware. PC-RW Transactions have to wait in the middleware for their writesets before they can return to the user, and hence, perform worse than PC-LW transactions. In a 5-site system with evenly distributed workload, we can expect 20% of transactions to have PC-LW response times, and 80% to have PC-RW times. For LC only if there are two concurrent transactions that conflict, one has to wait for the writeset of the other to arrive, otherwise no wait is needed. Hence, the average response time is between PC-LW (do not wait for a writeset) and PC-RW (always wait for the writeset). At low load (10 tps) LC's response time is very close to PC-LW because there are barely any concurrent transactions in the system. Hence very few transactions have to wait in the middleware (very thin white stripe in the figure). At higher load (20 tps), more transactions reside concurrently in the system, hence, more conflicts occur and more transaction have to wait (larger white stripe in the figure).

As a summary, while the write set approach boosts performance in a LAN it does not in a WAN due to the additional message which leads to more contention in the GCS and additional waiting times for transactions to commit. If it is needed due to non-determinism, the local approach seems favorable over the primary copy approach.

4.4 Communication Choices

Delivery Alternatives Fig. 6 shows the response times when using agreed, safe, or optsafe delivery in Spread. We use 5 sites, Sym, and 100% updates. For optsafe delivery, we also show the early execution variant Esym. Results for PC and LC are not shown since their relative behavior to Sym is similar as above for all delivery types.

Agreed Sym performs by far the best due to the fast delivery of reliable messages compared to uniform reliable messages. Safe Sym is worst since the full delay of safe delivery is added to the response time. Optsafe Sym will start to execute upon agreed delivery but has to wait for the safe delivery to commit, leading to better response time than Safe Sym. However, the difference is very small since transaction execution only takes small percent of response time as shown in Fig. 5. Optsafe ESym is significantly better than Optsafe Sym since it allows for more concurrent execution. Still the execution time is determined by the final safe delivery.

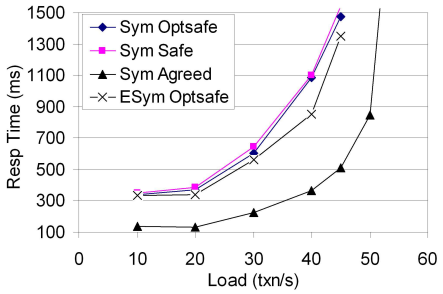

Fig. 6. Agreed, safe and optsafe delivery

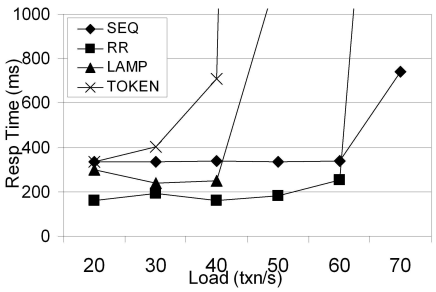

Fig. 7. Total Order Implementations

These results show that best performance can be achieved if we can accept the loss of some transactions in failure cases. However, if full atomicity is needed, optimistic delivery and especially our early execution variation can increase performance.

Total Order Implementations So far, we have used Spread which is a *token* based total order algorithm. In this section, we analyze different total order algorithms and their impact on our replicated database system using JavaGroups. Fig. 7 shows the response times for Sym, 100% updates, and 5 sites. TOKEN refers to the token based approach, SEQ to the sequencer based approach, LAMP to Lamport's algorithm, and RR to the simple round-robin algorithm. TOKEN has by far the worst performance due to the delay incurred by rotating the token. Using SEQ, despite requiring three messages per application message, and the potential bottleneck of one sequencer site, offers better performance than TOKEN. Furthermore it leads to stable response times until the sequencer becomes saturated at around 70 tps. LAMP provides faster response times than TOKEN and SEQ for low loads up to 40 tps although this protocol provides safe delivery while the others only provide agreed delivery. Response times at 30 tps are better than at 20 tps because when more messages are sent, acks arrive faster. However, LAMP saturates earlier due to CPU saturation since it has to keep track of acks. RR has the lowest response time of all since there are no additional messages and message rounds. It saturates only shortly before the sequencer due to CPU overhead. As such, RR seems the preferable choice if agreed delivery is sufficient.

In order to understand the impact of the application on the performance of the GCS we also conducted experiments with JavaGroups and a light-weight message generator client, without the replicated database. The results are similar but generally larger message throughput is achievable because there is no serious application competing with JavaGroups for resources. An interesting exception was SEQ that achieves a far higher throughput than the others without database application, but only a slightly higher throughput when run together with our replicated database system. This is because the sequencer site is saturated much earlier when resources are consumed by the database application. Looking at Spread's agreed token based algorithm from Fig. 6 and JavaGroup's TOKEN, transaction response time with JavaGroups are about 2 times higher than with Spread at the same load. This is probably due to implementation choices like the programming language and internal data structures within the GCS.

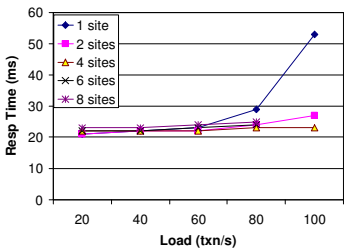

Fig. 8. Scalability: Queries

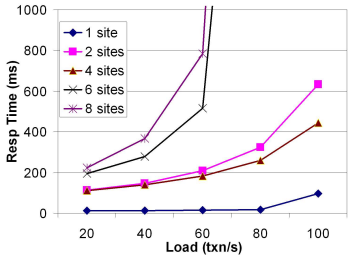

Fig. 9. Scalability: Update transactions

4.5 Scalability in a WAN

This experiment evaluates the performance for system configurations between one and 8 sites. This time, the workload has 50% update transactions and 50% queries. We use Spread's agreed delivery and the Sym algorithm.

Fig. 8 shows that the response times for queries are very similar for all system sizes for smaller loads. At 80 tps a 1-site system starts to saturate, and the response times for queries deteriorates while larger system sizes still provide good response times. With more sites, queries are better distributed and hence, each site is less loaded. Fig. 9 shows the response times for update transactions. A centralized system is always the best for updates since there is no communication. A 4-site system has consistently better performance than a 2-site system because the query load is distributed among more sites, and hence, response times are faster even for updates. For 6 and 8 sites, the response time increases because of the increased overhead to reach total order.

In summary, for a typical, read-intensive workload, a WAN replicated system of 8 sites can achieve an astonishingly high throughput while providing 1-copy-serializability. Queries are not affected by update transactions and can take advantage of distribution.

5 Conclusions

This paper presents a detailed WAN based performance analysis of group communication based data replication providing full data consistency. Since existing protocols developed for LANs have limitations in WANs due to the message overhead, we have proposed optimizations (local copy approach, early execution variant) that alleviate the problems. In general, we believe that consistent database replication is feasible.

Summarizing the alternatives, a symmetric approach is preferable over write set based approaches because of the lower message overhead. However, if non-determinism requires a writeset approach, our new local copy approach works better than a primary copy approach for the majority of transactions. Choosing agreed delivery over safe delivery can help to reduce response times. If the application requires atomicity in all situations, optsafe delivery, and especially the newly proposed early execution variation can alleviate the high message delay of safe delivery. The choice of total order algorithm and its implementation has an impact on the performance. An algorithm which simply delivers messages in round-robin appears to have good performance in WAN.

References

1. Y. Amir, C. Danilov, M. Miskin-Amir, J. Stanton, and C. Tutu. On the Performance of Consistent Wide-Area Database Replication. Technical Report CNDS-2003-3, CNDS, John Hopkins University, 2003.
2. Y. Amir and C. Tutu. From Total Order to Database Replication. In *Proc. of ICDCS*, 2002.
3. C. Amza, A. L. Cox, and W. Zwaenepoel. Conflict-Aware Scheduling for Dynamic Content Applications. In *USENIX Symp. on Internet Tech. and Sys.*, 2003.
4. T. Anderson, Y. Breitbart, H. F. Korth, and A. Wool. Replication, Consistency, and Practicality: Are These Mutually Exclusive? In *ACM SIGMOD Conf.*, 1998.
5. R. Baldoni, S. Cimmino, and C. Marchetti. Total Order Communications: A Practical Analysis. In *EDCC, Lecture Notes in Computer Science, Vol. 3463*, pages 38–54, Mar. 2005.
6. E. Cecchet, J. Marguerite, and W. Zwaenepoel. C-JDBC: Flexible database clustering middleware. In *USENIX Conference*, 2004.
7. G. V. Chockler, N. Huleihel, and D. Dolev. An Adaptive Totally Ordered Multicast Protocol that Tolerates Partitions. In *PODC*, 1998.
8. G. V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *ACM Computer Surveys*, 33(4), 2001.
9. X. Defago, A. Schiper, and P. Urban. Comparative Performance Analysis of Ordering Strategies in Atomic Broadcast Algorithms. *IEICE Trans. Inf. and Syst.*, E86-D(12), Dec. 2003.
10. E. Pacitti and T. Ozsu and C. Coulon. Preventive Multi-master Replication in a Cluster of Autonomous Databases. In *Euro-Par*, 2003.
11. U. Fritzke and P. Ingels. Transactions on Partially Replicated Data based on Reliable and Atomic Multicasts. In *Proc. of ICDCS*, 2001.
12. R. Goldring. A discussion of relational database replication technology. *InfoDB*, 8(1), 1994.
13. J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proc. of SIGMOD*, 1996.
14. J. Holliday, D. Agrawal, and A. E. Abbadi. The Performance of Database Replication with Group Communication. In *FTCS*, 1999.
15. Java Groups. homepage: http://www.jgroups.org/.
16. R. Jiménez-Peris, M. Patiño-Martínez, B. Kemme, and G. Alonso. Improving the Scalability of Fault-Tolerant Database Clusters. In *ICDCS*, 2002.
17. K. Böhm and T. Grabs and U. Röhm and H.J. Schek. Evaluating the Coordination Overhead of Replica Maintenance in a Cluster of Databases. In *Proc. of Euro-Par*, 2000.
18. B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing Transactions over Optimistic Atomic Broadcast Protocols. In *ICDCS*, 1999.
19. L. Rodrigues and P. Vicente. An Indulgent Uniform Total Order Algorithm with Optimistic Delivery. In *SRDS*, 2002.
20. M. Patio-Martnez, R. Jimnez-Peris, B. Kemme, and G. Alonso. Scalable Replication in Database Clusters. In *DISC'00*, pages 315–329, Toledo, Spain, 2000. Springer.
21. F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distributed and Parallel Databases*, 14(1), 2003.
22. C. Plattner and G. Alonso. Ganymed: Scalable replication for transactional web applications. In *Middleware*, 2004.
23. L. Rodrigues, H. Miranda, R. Almeida, J. Martins, and P. Vicente. Strong Replication in the GlobData Middleware. In *Workshop on Dependable Middleware-Based Systems*, 2002.
24. M. Shapiro and Y. Saito. Scaling optimistic replication. In *Future Directions in Distributed Computing*. Springer LNCS 2584, 2003.
25. A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic Total Order in Wide Area Networks. In *SRDS*, 2002.
26. Spread. homepage: http://www.spread.org/.