Data Replication in a Distributed System: A Performance Study

San-Yih Hwang[1] and Keith K. S. Lee[2] and Y. H. Chin[2*]

[1] Department of Information Management, National Sun Yat-Sen University, Kaohsiung 80424, Taiwan
syhwang@mis.nsysu.edu.tw

[2] Department of Computer Science, National Tsing Hua University, Hsin-Chu 30043, Taiwan
yhchin@cs.nthu.edu.tw

Abstract. In this paper we investigate the performance issues of data replication in a loosely coupled distributed database system, where a set of database servers are connected via a network. A database replication scheme, *Replication with Divergence*, which allows some degree of divergence between the primary and the secondary copies of the same data object, is compared to other two schemes that, respectively, disallows replication and maintains all replicated copies consistent at all times. The impact of some tunable factors, such as cache size and the update propagation probability, on the performance of *Replication with Divergence* is also investigated. These results shed light on the performance issues that were not addressed in previous studies on replication of distributed database systems.

1 Introduction

Today's international enterprises typically own tens or hundreds of servers distributed around the world. Data Replication is an important technique for improving performance of applications in such organizations. Due to the geographical dispersion, demanding multiple copies of the same data to agree on a single value imposes severe problems on performance and, in most cases, is an overkill. Thus, many distributed organizations today choose to have local access to a copy of global information rather than global access to centrally stored information [9]. Synchronization among the replicated data is done in a predefined schedule. Facing the increasing need of replication for the enterprised or interenterprised applications, many vendors, including Oracle, Informix, Sybase, Lotus and Microsoft, have offered products that support data replication (e.g., Oracle 7, Sybase Replication Server, Informix 6, Lotus Notes and MS Exchange).

A typical computer environment in a distributed enterprise consists of a set of database servers interconnected via a (wide-area) network to provide aggregate database services. Each server is capable of storing and manipulating a limited

* To whom all correspondence should be addressed.

amount of data. However, a user can issue a query or an update request based on the entire data set to any server. If the required data resides entirely on the database server, this request may be completely served locally. Otherwise it has to be shipped to other servers for processing. We call such a system *loosely coupled distributed database system* since each database server is able to operate independently.

In this paper, we conduct a performance study on various schemes for the data access on a loosely coupled distributed database system. We aim to understand the performance behavior of various replication schemes under different operating regions and identify the key factors that affect their relative performance. Specifically, we consider the following three schemes for handling data requests in a loosely coupled distributed database system.

1. *Disjoint Data Partition*: This scheme disallows any data replication. That is, each data item is stored on exactly one server.
2. *Replication with Consistency*: In this scheme, each server stores data to its maximum extent. A data item can be stored by more than one server. Besides, all copies of a data item are consistent on one value at any time. Thus, some synchronous protocol such as read-one-write-all must be employed to ensure consistency.
3. *Replication with Divergence*: Similar to the previous scheme, this scheme also allows data replication. However, all copies of a data item are distinguished as *primary* and *secondary*. A data item has exactly one primary copy and several secondary copies. While the value of the primary copy must reflect the most recent update, values of secondary copies may diverge to different degrees, as required and specified by relevant users.
   There are various types of divergence specifications for coherence conditions proposed in the literature [1, 8, 6]. These specifications specify the allowed difference between the primary copy and some secondary copies of the same object in terms of value, time lag, and others.

Our performance study shows that *Replication with Divergence* always surpasses *Replication with Consistency*, and it outperforms *Disjoint Data Partition* for read operations and is comparable to *Disjoint Data Partition* for write operations when contention is not high. When contention is extremely high, the best strategy is not to cache any data items. That is, *Disjoint Data Partition* performs the best for both read and write operations under this circumstance.

Key factors that affect the performance of *Replication with Divergence* are also examined. We intend to understand what the system administrator can do to further improve the performance. Specifically, we seeks answers to the following two questions:

- Is it worthwhile to add more storage space to each database server (so that more data items can be cached)?
- Should we maintain finer granularity on coherence conditions specified by the users, e.g., individual fields rather than entire relations, so as to eliminate unnecessary data propagation?

The analysis results show that increasing storage at each database server will benefit performance only when the system is under low to median contention. Besides, there is a tradeoff between the propagation probability and the cost for checking coherence constraints. The best performance occurs at a median propagation probability, where the communication traffic is effectively reduced, and the incurred coherence checking overhead is still reasonable.

The concept of applying weaker coherence constraints on an information retrieval system (IRS) was proposed in [1]. Their research conducted performance study and discussed implementation issues in a fairly complete manner. However, the reference architecture they assumed in their study is *Information Retrieval Systems*, which is very different from ours, *Loosely Coupled Distributed Database Systems*. Unlike an information retrieval system, a loosely coupled distributed database system consists of a set of peer database servers to provide aggregate database services. Thus, their results cannot be applied directly to our environment.

In [2], Barbara et al. proposed a demarcation protocol that is able to decompose a global constraint into a set of local ones such that a local transaction does not have to check the values of remote data. While their goal to locally complete a data request is the same as ours, the focus of their research is on decomposing and maintaining *global integrity constraints*, rather than temporal constraints, which is the main focus of our work.

Asynchronous propagation of an update request has also been employed in a distributed system to enhance availability and performance by several researchers [7, 3]. These approaches first commit a transaction at the primary site, and then gradually propagate it to other secondary sites. It has been shown that the lack of synchronization protocols (e.g., 2 phase commit) improves system performance and enables the system to continue in operation in the presence of failure. Our work not only utilizes asynchronous propagation but also employs selective propagation to further enhance system performance.

The rest of this paper is organized as follows. In Section 2, we present the performance model and conduct an analysis on the three schemes described previously. The analysis results are discussed in Section 3. Finally Section 4 concludes this paper.

2 Performance Model and Analysis

Assume there are m database servers interconnected by a network. Each server stores a subset of data that is available to users. A data request issued to a site can be either served by the server at that site or by the server(s) at some remote site(s). The system parameters and settings are summarized in Table 1.

For *Replication with Divergence* scheme, the data stored on each server can be partitioned into two kinds: primary and secondary. The primary copy is the main copy of the associated data item. Each data item has exactly a primary copy stored at one server and $k-1$ secondary copies stored at various servers. For a data request issued to a local server, the probability that this request is on the primary data stored on the local server is P_{lp} and the probability on the secondary data stored on the local server is P_{ls}. Thus the probability that this request is on the data not stored on the local server is $1 - P_{lp} - P_{ls}$. Note that the secondary data stored on a server are actually the data cached by the server. Thus, each unit of secondary data should have higher probability of being accessed by a local request than that of *non-cached* data. To better model the behavior of data access locality, we relate P_{ls} to k and m by $\sqrt{\frac{k-1}{m-1}}(1 - P_{lp})$. This function enables smaller k to yield bigger P_{ls}. For example, when $k \approx \frac{m}{4}$ (that is, the cached data at each site is about a quarter of the total data set), the probabilities for accessing secondary data and non-cached data are approximately equal (i.e., $\frac{1}{2}(1-P_{lp})$). Similarly, we can calculate the probabilities of a request accessing data stored on the local server for the other two schemes. For *Disjoint Data Partition* scheme, it is P_{lp}. For *Replication with Consistency*, it is $P_{lp} + P_{ls}$.

P_r denotes the probability that a data request is "read". P_p is used by *Replication with Divergence* to represent the probability that a write operation has to be propagated to a server storing a secondary copy to refresh its value. Note that the write operation is propagated to a site with a secondary copy only when the associated coherence constraint is violated. Each server is assumed to be a M/G/1 system and employs *round robin* scheduling. The average arrival rate to a local site is denoted as λ, and the average service rate of a server is μ.

The parameter settings in Table 1 are based on a system consisting of a median number of servers connected via a wide area network. In such a system, most operations are read (P_r is set to 80%) and most data required by an operation can be found in the local server ($P_{lp} + P_{ls}$ is set to be about 80%). A database server is able to serve hundreds of data requests per second. The external arrival rate to a server is ranged from low (10) to high (150) to model the effect of different levels of resource contention.

Now we are ready to derive some properties about the data requests in this environment. Let λ_i be the total (internal plus external) arrival rate of requests to the i'th server, μ_i be the mean service rate of the i'th server[3], and $p_i(k)$ be the probability that the i'th server has k requests queued. It has been shown that if the server is a M/G/1 system and employs *round robin* scheduling, p_i behaves as in a M/M/1 model [5]. That is,

$$p_i(k) = (1 - \rho_i)\rho_i^k,$$

where $\rho_i = \frac{\lambda_i}{\mu_i}$. Thus, the mean response time (T_i) of a request submitted to the i'th server is $\frac{1}{\mu_i - \lambda_i}$.

To simplify the study, we consider a homogeneous environment, in which each server has equivalent power, and each local site has the same arrival rate. Thus $\mu_i = \mu$, $1 \leq i \leq n$. Let λ denote the external arrival rate of requests to a local site. We are now interested in finding the value of λ_i, $1 \leq i \leq n$ (i.e., the total

[3] We assume both read and write requests take the same average service time, i.e., $\frac{1}{\mu_i}$ at the i'th server.

arrival rate, including both local and remote requests, to a server). Note that a request may be served by the local server and/or some remote servers. Since the considered environment is homogeneous, the total arrival rate to a server is equal to the rate of total services introduced by local requests at a site.

1. *Disjoint Data Partition*: Since an arriving request will be either served by the local server or by exactly one remote server in this scheme, the total arrival rate to a server is λ.
2. *Replication with Consistency*: There are various strategies for realizing this scheme, e.g., *read one write all* and *voting* [4]. In this work, we assume this scheme employs *read one write all* strategy since it is commonly used and has better performance for read operations. In this case, a read operation can be served by exactly one server, either locally or remotely, and a write operation must be served by k servers, where k is the number of copies a data item has. Thus, the total arrival rate to a server is $\lambda(P_r + (1 - P_r)k)$.
3. *Replication with Divergence*: In this scheme, each read request will be served by exactly one server. A write request is first served by the server containing the primary copy. After that, a constraint will be checked to decide whether or not to propagate this request to another server containing a secondary copy of the data item. If so, this request is propagated to that server. Note that there are $k - 1$ servers containing secondary copies of a data item, and the probability that an update needs to be propagated to a secondary copy is P_p. Thus the total arrival rate to a server is $\lambda(1 + P_p(k - 1)(1 - P_r))$.

Let T_{dj} be the average turn around time of a request serviced by a server under *Disjoint Data Partition*, T_{rc} be that under *Replication with Consistency*, and T_{rd} be that under *Replication with Divergence*. The following equations are clear from the previous discussions [5]:

$$T_{dj} = \frac{1}{\mu - \lambda},\ T_{rc} = \frac{1}{\mu - \lambda(P_r + (1 - P_r)k)},\ T_{rd} = \frac{1}{\mu - \lambda(1 + P_p(k - 1)(1 - P_r))}$$

A request submitted to a local site will be served by either the local server or some remote server(s), depending on the scheme employed. Considering the accessing probabilities to different regions of data (P_{lp}, P_{ls}, and $1 - P_{lp} - P_{ls}$) and the communication overhead ($\$_{dj}$, $\$_{rc}$ and $\$_{rd}$ for *Disjoint Data Partition*, *Replication with Consistency*, and *Replication with Divergence* respectively), we are able to derive the average request response time for both read and write operations. The results are shown in Table 2. For *Disjoint Data Partition*, there is $1 - P_{lp}$ probability that the required data of a request, which is either read or write, resides on another server. For a read request under *Replication with Consistency* or *Replication with Divergence*, the probability that the demanded data is not cached is $1 - P_{lp} - P_{ls}$. A write request under *Replication with Consistency* must incur the delay of **one** round of message passing to update all copies of the specified object. For *Replication with Divergence*, there is $1 - P_{lp}$ probability that a write request has to be sent to the server containing the primary copy of the specified object. Note that in this case, the write request is reported successfully as soon as the primary copy is updated; the update of secondary copies, if any, is conducted off-line.

3 Analytical Results

In this section, we show the performance of various schemes under various operating regions. The primary performance metric we adopt is *average request response time*. Besides we vary *arrival rate* in each experiment to observe the behavior of different schemes under different levels of resource contention.

Due to space limitation, we present two sets of experiments in this paper. The first set of experiments compares the three schemes, namely *Disjoint Data Partition*, *Replication with Consistency*, and *Replication with Divergence* under default settings, where neither network contention nor coherence checking overhead is modeled.The second set of experiments is intended to see how tunable parameters, namely the size of storage at each server and the update propagation probability, affect the performance of *Replication with Divergence*. In the investigation of update propagation probability, coherence checking overhead is modeled.

3.1 Experiment Set 1

This set of experiments compares *Replication with Divergence* with the other two schemes in terms of the mean response time of a request. We intend to know under what circumstances *Replication with Divergence* will perform better. We assume a constant time $ for transmitting a packet on networks.

We first compare the average read response time of *Replication with Divergence* with that of *Replication with Consistency*. Referring to Table 2, it is clear that these two schemes incur the same amount of communication time for a read request. With respect to the service turnaround time, T_{rd} is less than T_{rc}, and the difference becomes more significant as the request arrival rate increases. We next compare the average read response time of *Replication with Divergence* with that of *Disjoint Data Partition*. While *Replication with Divergence* incurs less communication time, it introduces more turn around time for a request(T_{rd}), compared to *Disjoint Data Partition* scheme. As contention becomes higher (i.e., more requests arrive), the weight of service turn around time gradually outperforms that of communication overhead. The above observations can be seen visualized in Figure 1.

We can do similar analysis for the average response time of write operations. The average write response time is also depicted in Figure 1. It can be seen that *Replication with Divergence* is always better than *Replication with Consistency*. The performance of *Replication with Divergence* is comparable to that of *Disjoint Data Partition* except when the system is under very high contention, where the difference becomes significant. Overall, we conclude that *Replication with Divergence* is an attractive approach under most operating regions in which it benefits a lot for read operations and pays only a little price for write operations. When the system is under high contention, *Disjoint Data Partition* scheme becomes the best choice.

3.2 Experiment Set 2

In this experiment set, we intend to observe how the *key* factors affect the performance of *Replication with Divergence*. Specifically, we examine the impact of k, the number of copies a data item has, and P_p, the probability of propagating a write operation to a secondary copy, on the performance. Note that the storage capacity in each server affects the value of k. P_p is decided by the number and complexity of coherence constraints. To reduce P_p, more and more complicated constraints need to be checked.

We first investigate the effect of k on the performance of both read and write operations. The mean response time for both types of operations versus request arrival rates are depicted in Figure 2. As expected, bigger k value has negative impact on the performance of write requests, since more replication implies more update propagation. With respect to the performance of read requests, bigger k value, which implies that more data are replicated, does not always perform better. This is because, while bigger k increases the probability of local access, it also causes higher system load.

The next experiment examines the effect of P_p. We model the cost of checking coherence constraints by the following function:

$$costofcheckingcoherenceconstraints = \frac{C}{\mu} \times (\frac{1 - P_p}{0.75})^2,$$

where $0 \leq C \leq 1$, and $0.25 \leq P_p \leq 1$.

Figure 3 shows the response time of read and write requests under various P_p when $C = 0$ (i.e., no cost for coherence constraint checking). As expected, lower P_p has better system performance for both read and write operations. When coherence constraint checking is taken into account, lower P_p is not necessarily better. Figure 4 shows the response time of read and write requests when $C = 1$. In this case, the best performance occurs when $P_p = 0.5$ or 0.75. This is because a median P_p implies that the propagation of update operations has been effectively reduced, while the cost of coherence constraint checking is only slightly increased.

In summary, this set of experiments indicates that increasing the server storage capacity (so as to cache more data) and/or decreasing the write propagation probability (while conducting more complicated constraint checking) do not always benefit performance. Only when system contention is low, increasing the server storage capacity increases performance. Besides, the amount of coherence checking work should be constrained within a median range to achieve peak performance.

We have also examined the effect of network contention on the three schemes. The examination showed that network contention will further deteriorate the performance of *Replication with Consistency* but impose insignificant implication on the relative performance between *Replication with Divergence* and *Disjoint Data Partition*.

4 Concluding Remarks

We have analyzed the performance of various schemes for data replication and investigated their implementation issues in a loosely coupled distributed database system, where a set of database servers are clustered to provide aggregate database services. Three schemes, namely *Replication with Divergence*, *Replication with Consistency* and *Disjoint Data Partition*, are compared. While *Replication with Consistency* and *Disjoint Data Partition* maintain a single logical value for each data item, *Replication with Divergence* allows users that access the same data object from different sites to perceive values with some (bounded) degree of divergence. We have shown that *Replication with Divergence* outperforms *Replication with Consistency* under all circumstances and has better performance than *Disjoint Data Partition* in most operating regions except when the system is under very high resource contention. We have also investigated the impact of tunable factors on the performance of *Replication with Divergence*. It has been shown that the best performance of *Replication with Divergence* occurs at a median probability of update propagation, where the overhead for checking coherence constraints is not too much.

References

1. R. Alonso, D. Barbar, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transaction on Database Systems*, 15(3):359–384, 1990.
2. D. Barbara-Milla and H. Garcia-Molina. The demarcation protocol: A technique for maintaining constraints in distributed database systems. *The International Journal on Very Large Data Bases*, 3(3):325–355, 1994.
3. A. Bhide, A. Goyal, H. I. Hsiao, and Anant Jhingran. An efficient scheme for providing high availability. In *Proc. of ACM SIGMOD Int'l. Conf. on Management of Data*, pages 236–245, 1992.
4. S. B. Davidson. Replicated data and partition failures. In S. Mullender, editor, *Distributed Systems*, pages 265–292. Addison Wesley, 1989.
5. L. Kleinrock. *Queueing Systems, Vol. 1: Theory.* Wiley-Innterscience, New York, 1975.
6. R. Lenz, T. Kirsche, and B. Reinwald. Aspect – specifying consistency requirements for replicated data. In *Proc. of the Seventh International Conference on Parallel and Distributed Computing Systems*, 1994.
7. C. A. Polyzois and H. Garcia-Molina. Evaluation of remote backup algorithms for transaction processing systems. *ACM Transaction on Database Systems*, 19(3):423–449, 1994.
8. M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer*, Dec. 1991.
9. D. Yavin. Replication's fast track. *Bytes*, Aug. 1995.

This article was processed using the LaTeX macro package with LLNCS style

Table 1. System Parameters and Settings

Parameter	Meaning	Setting
m	Number of servers in the system	10
k	Number of copies a data item has	4
P_{lp}	Probability of a local request to the primary copy	0.5
P_{ls}	Probability of a local request to the secondary copy	$\sqrt{\frac{k-1}{m-1}}(1-P_{lp})$
P_r	Probability of a read request	0.8
P_p	Probability of propagating a write operation to a secondary copy	0.25
$\$$	Communication delay for transmitting a packet	0.01 sec.
$\frac{1}{\mu}$	Mean service time of a read or write request	0.005 sec.
λ	Mean arrival rate of requests to a local site	10 ... 150/sec.

Table 2. Average Response Time of a Request

Request	Disjoint	Replication with Consistency	Replication with Divergence
read	$T_{dj} + 2\$_{dj}(1-P_{lp})$	$T_{rc} + 2\$_{rc}(1-P_{lp}-P_{ls})$	$T_{rd} + 2\$_{rd}(1-P_{lp}-P_{ls})$
write	$T_{dj} + 2\$_{dj}(1-P_{lp})$	$T_{rc} + 2\$_{rc}$	$T_{rd} + 2\$_{rd}(1-P_{lp})$

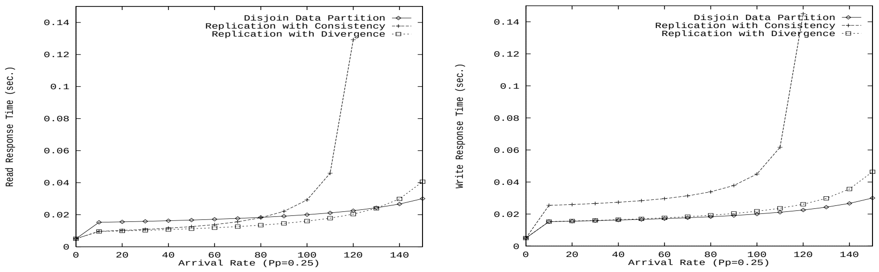

Fig. 1. Average Response Time of a Read/Write Request

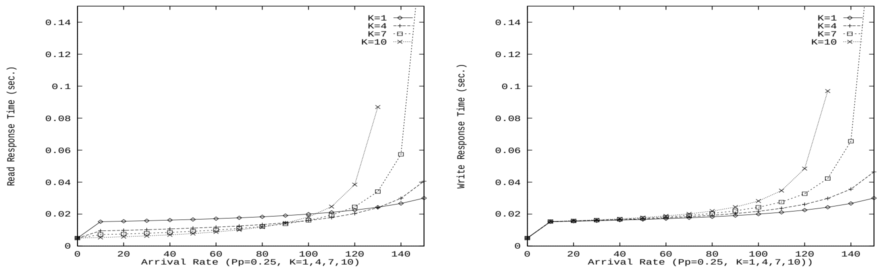

Fig. 2. Read/Write Response Time for Different k Values

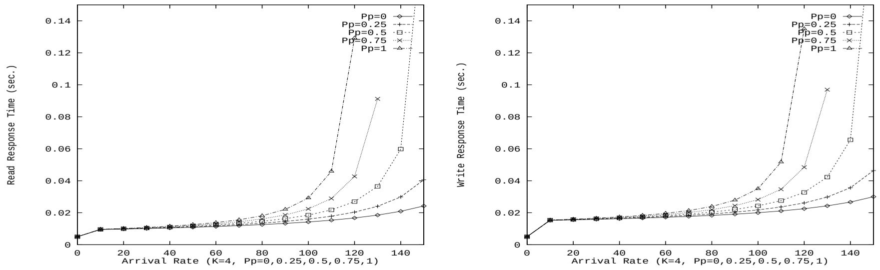

Fig. 3. Read/Write Response Time for Different P_p without Constraint Checking Overhead

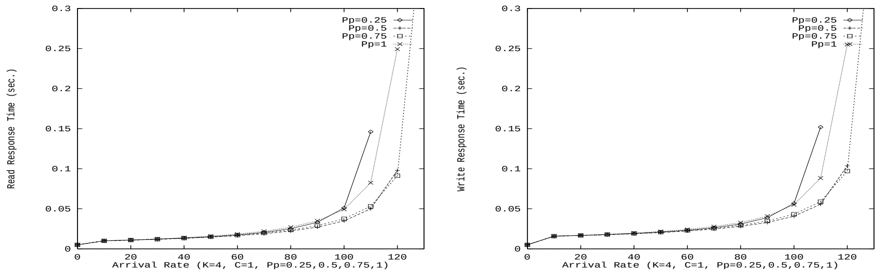

Fig. 4. Read/Write Response Time for Different P_p with Constraint Checking Overhead