What Price Replication?

M. L. Liu D. Agrawal A. El Abbadi
Department of Computer Science
University of California
Santa Barbara, CA 93106
May 25, 1994

Abstract

Replicated data is employed in distributed databases to enhance data availability. However, the benefit of data availability is only realized at the cost of elaborate algorithms which hide the underlying complexity of maintaining multiple copies of a single data item. The concern about the performance impact of replica control is part of the reasons why replication, although extensively researched, has yet to receive wide acceptance in practice. This paper makes use of a simulation model to explore the performance tradeoffs of data replication.

Keywords: replication control, transaction response time, throughput, data availability.

1 INTRODUCTION

Replicated data is employed in distributed databases to enhance data availability: multiple copies of a critical data item are maintained, typically on separate sites, so that the data item can be retrieved even if some copies of the data item cannot be accessed due to system failures. However, this benefit of data availability is only realized at the cost of elaborate algorithms that hide the underlying complexity of maintaining multiple copies of a single data item. Ideally, an application should not be cognizant of the existence of replicated data, so that it is written as if for a non-replicated database. This criteria is relatively easy to achieve during normal system operation, but is much more difficult to cope with in the face of system failures, the existence of which is the main motivation for replicating data in the first place. The difficulty lies in keeping the copies consistent with each other in the face of system failures while at the same time maximizing the data availability. The algorithms which address these problems are called replica control algorithms [4].

Although replica control has been the subject of intensive research for quite some time now, it has yet to fulfill its promise in practical applications. In the current state of distributed database technology, data replication, if implemented at all, is typically enforced by the *read-one, write all* protocol. More complicated, less restrictive replica control protocols, though a popular topic for research, are not implemented in any widely-used systems [26]. A major reason for this lack of acceptance is that the performance impact of these protocols cannot be easily quantified, as very little existing performance figures of commercial database systems which support copies are available. A natural question arises: Is replication worth it? In other words: Does replication truly enhance data availability? While the presence of multiple copies should increase the accessibility of data objects during failures, the maintenance of the copies itself requires significant overhead, in terms of messages and log-writes required by the replica-control protocols. It is conceivable that the operational overhead imposed by replica control is such that its performance impact mitigates or perhaps even override the benefits of providing accessible copies.

This paper investigates this question. A brief description of related previous work is presented in Section 2. Section 3 presents an overview of replica control protocols. We describe the write-all available approach, perhaps the most prevalent replica control protocol in practice, and the quorum consensus protocol, which we submit to be a more appealing algorithm. In section 4, we describe the simulation model which we employed to investigate the performance of these protocols against that of a non-replicated system in a client-server environment. Section 5 then presents the experiment results while Section 6 summarizes the results. We conclude the paper with our observations regarding the performance impact of replication in database systems.

2 Related Previous Work

Replica control is an ongoing subject of research. Most existing works are analytical in nature. However, there is a volume of work based on experimentation. Pu, Noe, and Proudfoot studied a technique of regeneration of replicated object [30] using an implementation(Eden). In [6], Bhargava, Noll, and Sabo observed the overhead of a replica control protocol and the effects of failures on data availability on the prototype system RAID. Simulation has also been used in a number of existing works. In [7] Carey and Livny compared the performances of concurrency control algorithms in replicated systems. Pâris, Long, and Glockner evaluated consistency algorithms for replicated files in [17]. Our work is in the same vein as the SETH experiment described in [5]. However, we differ in that our model contains details of concurrency control, atomic commitment, and two different replica control protocols, and our model simulates failures. Also, we are primarily interested in the tradeoff between traditional metrics of transaction performance (response time and throughput) and data availability rather than the performance of one particular protocol.

3 Replica Control Protocols

A *database* consists of a set of named *data items*. A user interacts with a database through the use of *transactions*. A transaction is an execution of a program which contains a sequence of data-accessing operations[4]. A *distributed database system* is a collection of sites each of which contains a database and the sites are connected by a communication network. In a non-replicated distributed database system, a transaction originates at a site, called the *coordinator*, which forwards each operation contained in the transaction to the appropriate database site where the target data item is stored. For our simulation model, each operation either *reads* from or *writes* to a data item. At the end of the execution of the operations, the coordinator issues a *commit* or an *abort* to all the participating sites to terminate the transaction. To preserve data integrity, it is necessary for all participating sites to consistently perform a single logical action, either commit or abort. This consistency is enforced by an *atomic commitment protocol* [4, 15]. In addition, *concurrency control* protocol, for example, *two-phase locking*, ensures that the interleaved execution of operations from different transactions is correct, that is, *serialized* [4, 15].

In a replicated database, a data item may be stored redundantly so that copies of the item are distributed on different sites. The coordinator translates each Read and Write on a data item in such a manner that the effect of the operation on the multiple copies of the object appears as though the object had a single copy. This property, known as *one-copy equivalence*, is the correctness criteria for transaction execution on a replicated database system [4].

On a system with replicated data, each Read(x) is translated by the *transaction coordinator* into Read(x_{A_1}), ..., Read(x_{A_m}), where $x_{A_1}$is the copy of data item x at site A_1. Similarly, each Write(x) is translated into Write(x_{B_1}), ..., Write(x_{B_n}) [4]. The composition of the read set $R = \{A_1, ..., A_m\}$ and the write set $W = \{B_1, ..., B_n\}$ depends on each protocol.

For clarification, we should point out that this paper does not deal with large-scale replication where there are a larger number of copies for each data object. Rather, we assumed that a limited number of copies are maintained for each replicated data object. Below we will discuss two main families of existing protocols for replica control in such an environment.

3.1 The Primary Copy Protocol

In the absence of failures, replica control can be easily achieved by assigning R, the read set, to contain any copy of data item x, and assigning W, the write set, to contain all copies of x. To minimize response time for read operations, R can be set to contain the copy of x that is nearest to the transaction coordinator.

With failures, however, writing all copies can cause indefinite blocking, which is unacceptable in practice. Hence the write-all approach is modified to write all copies available to the transaction coordinator. The most commonly known protocol of this genre is the *primary copy* protocol [3, 32]. The protocol is designed for systems where a primary copy of each data object is located at one site, while secondary copies of the object are distributed among other sites. A read operation is directed only to the primary copy, while a write operation results in updates of the primary copy as well as all available secondary copies. Unavailable secondary copies receive the update on a deferred basis. If a primary copy fails, a secondary copy is chosen to assume the role of the primary copy.

The primary-copy protocol is perhaps the most widely accepted replica control protocol in practice such as Distributed INGRES [33, 32], as well as in research prototypes such as the Harp Distributed File System [25]. The basic appeal of the protocol is in its conceptual simplicity, and, in environments where the primary copy can be chosen to be the closest to the transaction coordinator, the protocol yields the best response time for read operations. However, in some implementations, the primary copy protocol does not guarantee data consistency in the presence of network partitioning. This is due to the possibility that different partitions of a network may adopt inconsistent views of the designation of the primary copy for a given data object.

In other implementations, the protocol has been enhanced to tolerate network partitioning by maintaining a systemwide consistent view of the replica configuration. In INGRES [32], the configuration is maintained system-wide in a ‘ data structure called an ‘up list”. Based on the up list and a pre-defined ordering among the sites, a group of sites can consistently determine which of the copies of a data object is the current primary copy. On the other hand, the Harp distributed file system [25] uses the concept of *view changes*, first introduced in [10], to maintain a consistent view of the organization among each individual groups of copies. In general, an elaborate algorithm is required for the primary copy protocol to guarantee a consistent view of the replica configuration. The algorithm calls for a two-phased protocol similar to the *two-phase commit protocol* to attain consensus among the sites or copies regarding the state of the copy sites.

3.2 The Quorum Consensus Protocol

In the *quorum consensus* (QC) algorithm [14], a non-negative weight is assigned to each copy of a replicated object x. A read threshold RT and a write threshold WT are also defined for x such that both $2 * WT$ and $(RT + WT)$ exceed the total weight of all copies. A read quorum of x is any set of copies of x with a weight of at least RT; a write quorum of x is any set of copies of x with a weight of at least WT. By design, each write quorum of x has at least one copy in common with every read quorum or any other write quorum. A write operation to x, write(x), is translated into a set of write operations on each physical copy of *some* write quorum of x; while a read(x) is translated into a set of reads on each physical copy of *some* read quorum of x. In addition, a version number is maintained by each copy. With each read or write, the maximum version number returned by the copies in a quorum is the latest version number and the copy or copies associated with this version number contain(s) the most current value of x. For a write, all copies in the write quorum are updated and also assigned an incremented version number. As the weights assigned to the copies can be seen as "votes" from them, the algorithm is also known as the *Voting* algorithm. In its simplest form, a quorum is a majority set, in which case the algorithm is known as *Majority Voting* [31, 34].

QC is an elegant algorithm which provides one-copy-equivalence even in the presence of network partitioning. It does so without requiring complicated management of network configurations among the replica sites which is required for other replica control algorithms such as virtual partitions [11] or Primary Copy [32, 25]. It also does not require special treatment to recover copies, as outdated copies will not have the latest version number and so will not be read but will be over-written.

The QC algorithm has some well-known drawbacks, some of which that have been cited [4] are: (i) It requires multiple reads for each read; (ii) It requires a large number of copies ($2n + 1$) to tolerate n site failures; and (iii) All copies of x must be known in advance (to determine the weights and hence quorum configurations). Drawback (i) arises because, for each read operation, QC dispatches multiple read requests, and requires that a read quorum responds before the operation can be completed. In comparison, algorithms such as the Primary Copy require only the reading of one copy, which may be located on a site which minimizes the access time (that is, the primary copy is a local copy, or a copy at a site local to the transaction.) We argue that in a client-server environment (to be described) where the concept of local copies does not apply, the response time for the multiple reads, which take place concurrently, should not far exceed that for a single read, assuming that the sites are load-balanced and there is no excessive message processing latency. Drawback (ii) is true of all algorithms which addresses network partitioning. Finally, drawback (iii) has been addressed by some enhancements to QC which allow dynamic reconfiguration of the quorums.

As a replica control protocol, QC has received much attention from researchers [1, 9, 16, 18, 19, 29, 28, 27, 36, 20, 35] . Its implementation, however, is not widespread. The Majority Voting protocol is perhaps the version of the protocol that is typically considered.

4 Simulation Model

We made use of a simulation testbed to investigate the performance of the PC and QC protocols in a client-server environment. The model is a large, process-based simulation model implemented using MODSIM II, a high-level language which supports object-oriented programming and discrete-event simulation. Each site is an object, as is each transaction. Methods are defined for these objects to implement the functional modules of the model. Statistics for individual transactions are carried in the data fields of the objects for the transactions, and are accumulated using statistic procedures provided by MODSIM II. In our experiments, the means of the desired measurements are obtained by using the method of *batch means* [22, 13, 21]. The results reported in the paper are within the 90 percent confidence intervals for the quantities measured: the size of the confidence intervals of those measurements was within a few percent of the mean in all cases.

4.1 Overview of the Model

Our testbed models a loosely-coupled, distributed system consisting of a fixed number of database server sites and a fixed number of database client sites. Figure 1 illustrates the client-server environment for our model. Sites are connected by a communication network. Client sites operate independently of the server sites. For this study, the two sets of sites are mutually exclusive. Each site has non-sharable volatile storage as well as non-volatile storage (disks) and one or more CPUs. Each server manages and stores a subset of the database, known as a *database partition*. The database is a collection of files, which are replicated. The distribution of the

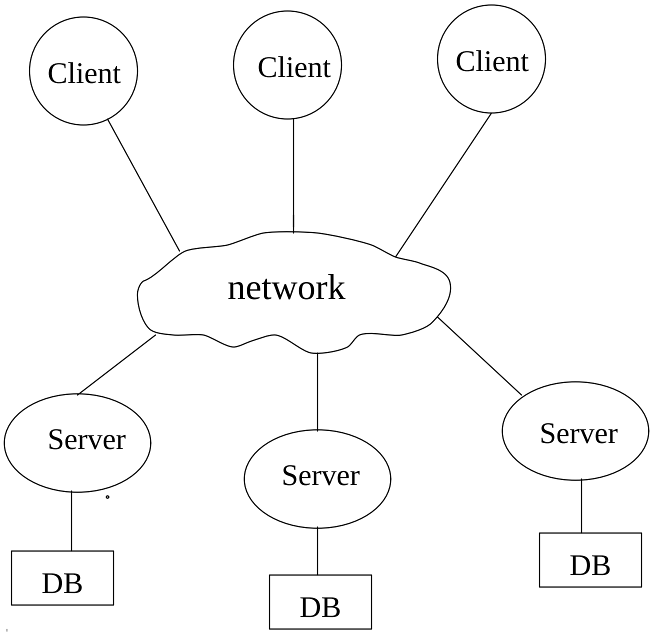

Figure 1: The Client-Server Environment

copies is known to all sites. Only the server with a file in its storage can directly access the file. Transactions emanate from a client site, which is then known as the *coordinator site* of the transaction. Each client site has disk storage (for logs) and one or more CPUs, but contains no data partition.

Figure 2 illustrates the modular components of our simulation model, which is based on the model described in [2]. Each module is briefly described as follows:

1. Transactions Generator: This is a global module which generates transactions and distributes them to the client sites. Transactions are generated independently of the processing at the client sites.

2. Site Failure Generator: This global module generates site failures and distributes them to the server sites. Failures are generated independently of the processing at each sites. A client site never fails.

3. The Communication Manager: This global module serves as a switch for routing messages from one site to another.

4. The sites: Each site is a module which in turn consists of the following components:

   (a) Transaction Manager: At each client site, this component is responsible for overseeing the execution of a transaction originating from that site. At each server site, this component is responsible for accepting transaction requests from the coordinator site.

   (b) Concurrency Control Manager: At each server site, this component provides concurrency control for transactions that have been submitted to the site.

   (c) Resource Manager: At each site, this component coordinates the consumption of site resources such as CPUs, disks, and log disk.

   (d) Failure Manager: At each server site, this component takes appropriate actions to effect a fail-stop failure upon receiving a site failure from the Site Failure Generator.

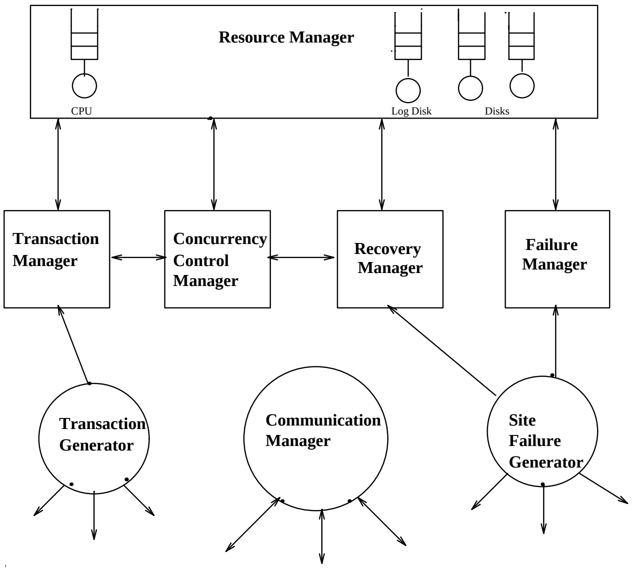

Figure 2: Modules of Simulation Model

(e) Recovery Manager: At each server site, this component coordinates the activities necessary to recover transactions interrupted by site failures. After-image logging is assumed for transaction recovery.

A detailed description of the model can be found in [24]. The next subsection provides an overview of the model's attributes.

4.2 Attributes of the Model

Choosing the parameter settings is a difficult issue in simulating a distributed database system. Due to the large number of parameters, it is not possible to vary all of them in our experiments. The choices of many of the parameters, such as the number of sites, the number of data objects, and the length of failures are limited by the scale of our model. Other parameters, such as the transaction size and failure rates are chosen to yield statistically significant results.

Following is a list of the attributes we chose for our model:

- Our model simulates 12 sites, four of which are client sites.
- There are 32 files of 200 pages each. Each file has three or five copies; the copies are evenly distributed among the server sites.
- Transactions are generated at a given rate independent of the sites, then assigned to the client sites using a uniform distribution.
- Transactions are evenly distributed between two classes: a read-only class and the other which contains write operations. Each transaction accesses 2 files on the average, and for each file accessed, one page is written to or read from.
- Each site has its own set of resources, including a CPU, a disk for data storage, and a log disk. Disk access time averages 0.02 time units. To simulate caching, a hit rate of 80% is assumed for disk accesses.

- Sites exchange messages at an overhead of 0.001 time units for message propagation, and 0.004 time units for message processing.

- At each site, strict two-phase locking is applied to individual pages for concurrency control [12, 4]. Deadlock avoidance is implemented by using a timeout interval based on the following heuristic [2] :

  $$TimeoutInterval = \mu(G) + k * \sigma(G)$$

  where $\mu(G)$ is the average lock request response time, $\sigma(G)$ is the standard deviation of the response time, and k is a weighting factor. Hence $TimeoutInterval$ is dynamically adjusted to reflect on-line estimation of lock request time. A transaction waiting for more than the timeout period is aborted.

- Communication timeout is set at 2.5 time units.

- The two-phase commit protocol is used for the atomic commitment of transactions.

- Server sites fail at a Poisson distribution rate of 0.02 (less frequent) or 0.5 (more frequent). Failure durations are uniformly distributed at 8 time units on the average. The failures are uniformly distributed among the server sites and are *fail-stop*: all activities at the failed site cease upon a failure. Key information needed for the recovery of the site is assumed to be retained in stable storage. Only site failures are simulated.

Note that the failure rate and failure duration are highly exaggerated. In real systems, mean time to failure and mean time to repair are expressed in terms of hours, which are not compatible with the time scale of our simulation model. We justify these parameter choices by the argument that our main interest lies in examining the performance of the protocols in the presence of failures, and that the exaggerated failures provide a microcosmic view of the protocols' behavior in the worst-case scenario.

4.3 Implementation of PC

For our experiments, the primary copies for the data objects (files) are uniformly distributed among the server sites. Thus, each server site in the model serves as the primary for multiple files as well as the secondary for other files. The replica configuration for each data item is assumed to be known to all active sites.

The deferred update scheme described in [32] is employed in our implementation. During the execution of a transaction, the coordinator forwards each operation to the primary site of the target data item. For a read operation, the value of the target data item is obtained from the primary site; there is no need to access any of the secondary copies. For a write operation, the update is performed at the primary site immediately, but updates at secondary sites are deferred: The updates made to all secondary copies at a site are collected and forwarded together to the site at commit time. Deferred update is possible for this protocol because the concurrency control on primary copies alone serves to ensure data consistency. Deferred updates potentially reduce the amount of message exchange required for each transaction, and therefore can be expected to benefit the performance of the transaction execution.

Each site is expected to maintain a list of the known status of other sites in terms of their accessibility. Such a list, which we term an *estimated uplist*, may be maintained by having the sites periodically exchanging status verification messages, as practiced by hosts on the Internet and on the Tandem non-stop system [9, 8]. In our model, we assume that such a list is available to the transaction coordinator with no additional overhead. If any site senses a failure among the server sites, a recovery process is initiated whereby data objects whose primary sites have failed may be reassigned new primary sites. If a secondary site is inaccessible to the coordinator, the failed replica is not updated. However, the update is logged to allow the failed copies to be updated upon the recovery of its host site. Our model assumes that such recovery actions take place in background mode.

For maintaining a consistent view of the replica configurations, we adopted the view-change algorithm outlined in [25] with the following stipulations: Whereas [25] envisions a view for the servers associated with each individual data item, we maintained one single view, that is, a global view, for all servers. (Note a view is different from the estimated uplist provided by the underlying system.) This approach is more suitable for our model, where the copies are distributed among a limited number of sites: If we were to maintain a consistent view for each replica group, there would be a significant overlap among the groups. The global view in fact corresponds to the uplist described in [32] and is used accordingly. For a given data item, the primary server is the server in the current uplist which has the lowest site number among the servers for the data item.

Although our model can be expanded for more than three servers per data item, to simplify the implementation we consider only three-member groups (one primary copy and two secondary copies).

4.4 Implementation of QC

While the basic concept of *quorum consensus* protocol is straightforward, its implementation is not trivial, especially when failures are taken into consideration. As mentioned earlier, the coordinator in QC translates each read and write on a data item x into reads and writes on multiple copies: a write(x) is translated into a set of writes on each copy of *some* write quorum of x; while a read(x) is translated into a set of reads on each copy of *some* read quorum of x. Note that for each read or write, it is sufficient to perform the operation on one particular quorum, although the operation can be performed on a superset of the quorum.

Although it is straightforward to define a quorum, the implementation for performing an operation on a quorum is less well defined. Although the basic idea is for the coordinator to dispatch an operation to a set of the copy sites such that responses are received from at least a quorum, it is not clear how this should be implemented. In [23], we describe four implementation approaches to gather a quorum for each operation. We start with the most naive approach of dispatching an operation to all copies, and subsequently propose alternative approaches, ending with the one which offers both efficiency and high data availability. In this approach, the coordinator selects a quorum set based on the estimated uplist introduced in the previous section. Assuming that the estimated uplist is reasonably current, each quorum selected by a coordinator will contain sites that are accessible. Consequently, the coordinator should be able to gather the necessary set of responses to render the operation successful even in the presence of failures. We assume that the estimated uplist is readily available to the coordinator as part of the status monitoring among the nodes already mentioned in the discussion of the implementation of PC. Note that we do not require that the uplist be accurate, hence the term "estimated uplist." Although it is desirable for the list to be as accurate as possible, the correctness of the execution of the transaction is enforced by the QC protocol itself.

In this study, we implement quorums as majority sets. Thus, for each operation a majority set is chosen using the approach described above, the operation is dispatched to the set, and, if all operations complete, the transaction is committed with the involvement of all server sites which have participated in its execution. Two sets of experiments were run under this protocol, involving three and five copies for each file respectively.

4.5 Performance Measurements

We measured the following quantities:

1. Average response time of committed transactions: This is a measurement of the elapsed time between when a transaction is initiated at its home site and when it is completed at the home site.

2. System throughput: The number of committed transactions per run divided by the time duration of the run.

3. Data availability: The number of committed operations over the total number of executed operations.

5 Simulation Results

For comparison, we ran experiments on our simulation testbed using (i) no replication (non-replicated), (ii) QC with three copies (QC-3), (iii) QC with five copies (QC-5), and (iv) PC with three copies (PC).

5.1 No Failure Case

We started by comparing the performance of the four replication approaches in the absence of failures. The overhead incurred by maintaining replicated data is obvious: For each operation in a transaction, additional processing may be required when compared with non-replicated data bases. In the case of PC, there is no additional overhead for read operations; but the effect of each write operation must be entered into a deferred update list. A significant overhead is also incurred during commit, when all participating sites (primary or secondary) must be involved in the interaction entailing message exchange and I/O accesses.

With QC, each operation, read or write, requires the interaction among the coordinator and a majority of the copies, thus there is a substantial amount of message and log overhead which is proportional to the size of the transaction. In addition, each operation incurs concurrency control overhead at each copy site contacted. As with PC, though to a lesser extent, the inclusion of the replica sites add to the overhead during the commit phase.

It should be noted that there exists at least one implementation of replicated data [25] which claims to outperform a non-replicated, conventional system in the absence of failures. This superiority in performance, however, is achieved by using a special underlying communication support (a reliable communication buffer, among other features) which replaces conventional event logging with a special message exchange facility. Our model assumes no such special support. In environments where the concept of "local copies" apply, it is possible for a replicated system using PC to outperform a non-replicated one, especially if read-only transactions dominate. This is because, under PC, a read needs to be performed on a local primary copy only, which, because of its proximity to the transaction coordinator, may have a shorter response time than reading from a replica at a remote site. However, since the concept of "local copies" does not apply in our model, this advantage of PC is not reflected in this study.

In our model, one can therefore expect the performance of a transaction for replicated data to compare unfavorably with that of a non-replicated data base system in the absence of failures. Figure 3 illustrates the throughput, response time, and the data availability when there are no failures in the system. The effect of the overheads introduced by the replica control protocols does not figure prominently until the transaction rate reaches beyond 25. Prior to that point the throughput, response time, and data availability are roughly equal in all cases. Under QC-5, thrashing occurs when the CPUs become overloaded at transaction rate = 25, whereas QC-3 and PC reaches their thrashing points at about 30. On the other hand, the non-replicated system does not reach thrashing point until transaction rate reaches 40. As the transaction rate increases, the non-replicated system enjoys the lowest response time, while PC also shows better response time compared to QC. The low response time of the non-replicated system is a direct result of its lower message overhead, while the better response time of PC in contrast to QC is attributable to the reduced overhead due to the use of deferred update under PC.

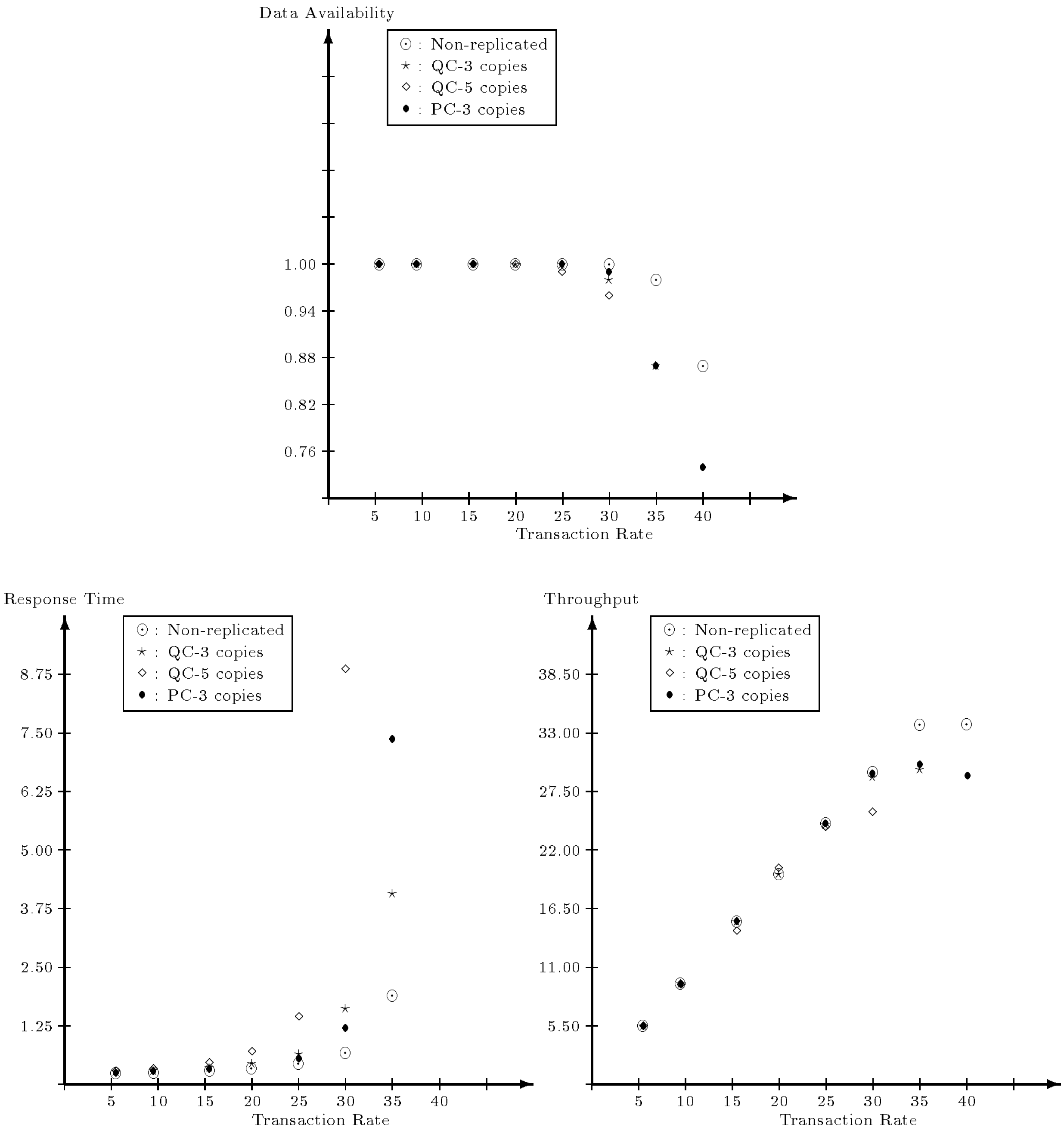

Figure 3: The Effect of Replication, No Failure

Theoretically, the data availability should be 100% at all times in the absence of failures. In our experiments, however, data availability starts to drop at thrashing points. The decline occurs because, as the CPU on each node becomes increasingly congested, more and more transactions are aborted due to timeout (while waiting for a response), causing a decrease in the ratio of committed operations over total operations.

This set of results indicate that in an environment such as the one that we modeled, the normal performance of PC is not significantly superior to the of QC-3. Also, until the thrashing point is reached, a replicated system performs no worse than a non-replicated system. However, the overhead required for data replication does have the effect of causing a system to reach thrashing point sooner than if there is no data replication.

In comparing the two QC protocols, one notes that QC-5's performance is comparable to that of QC-3 but quickly drops off as transaction rate reaches 23. The disparity is attributable to the larger message overhead required when more copies are involved in the execution of each operation, which leads to the CPUs becoming saturated at an earlier point in the case of QC-5.

5.2 Infrequent Failure Case

To justify the use of replication, the increased response time when there are no failures should be compensated by a corresponding gain in data availability in the presence of failures. In the case of QC, this presumed gain in data availability does not come at the expense of any overhead, as no special processing is necessary for error processing nor for recovery. With PC, however, there is an overhead in the view-change process, which takes place whenever a failure occurs and again whenever a site recovers (the view-change process is triggered by a change in the estimated uplist.) The view-change process is message intensive, as it involves many rounds of message exchange among the sites. Moreover, these messages are processed at a high priority to expedite the view change. As a result, one can expect a significant slowdown in the processing of other messages during such periods, including messages related to the execution of transactions. Hence the overhead of the view-change process can be expected to negatively affect the performance of transaction processing.

For our next group of experiments, we set fthe ailure rate to 0.05 per time unit (averaging two failures per run), with average failure length set to 8 time units. Figure 4 shows the results. The response time and throughput show a similar pattern as with the previous experiment, when no failures were introduced: The throughput and response times of the replicated cases are comparable to those of the non-replicated case until thrash points are reached. Owing to the failures now present, there is a slight decline in the throughput under all protocols. The benefit of data replication under QC is here reflected in the data availability: Both QC-3 and QC-5 enjoy a higher data availability than the non-replicated case until the transaction rate = 20 is reached. The benefit of PC is less obvious, as its data availability appears to be no better than that obtained with no replication. The explanation for this observation is that under PC, the gain in commitable operations during the brief failures is offset by the increase in aborted operations during the view-change processes.

5.3 Frequent Failure Case

While the previous set of experiments show that some increase in data availability is achieved in the presence of infrequent failures, we would expect the benefit to increase when failures are introduced more frequently. We increased the failure rate from 0.05 to 0.2 and repeated the experiments. The results are shown in Figure 5. With the highly exaggerated failure rate, almost one or more failures are present throughout the duration of a run. Under these circumstances, the benefit of replicating the data is highlighted by the better data availability and throughput under QC-3, QC-5, and PC (until they reach their thrash points). However, their average response times are higher, with PC being the lowest among the three: this is can be explained by the observation that

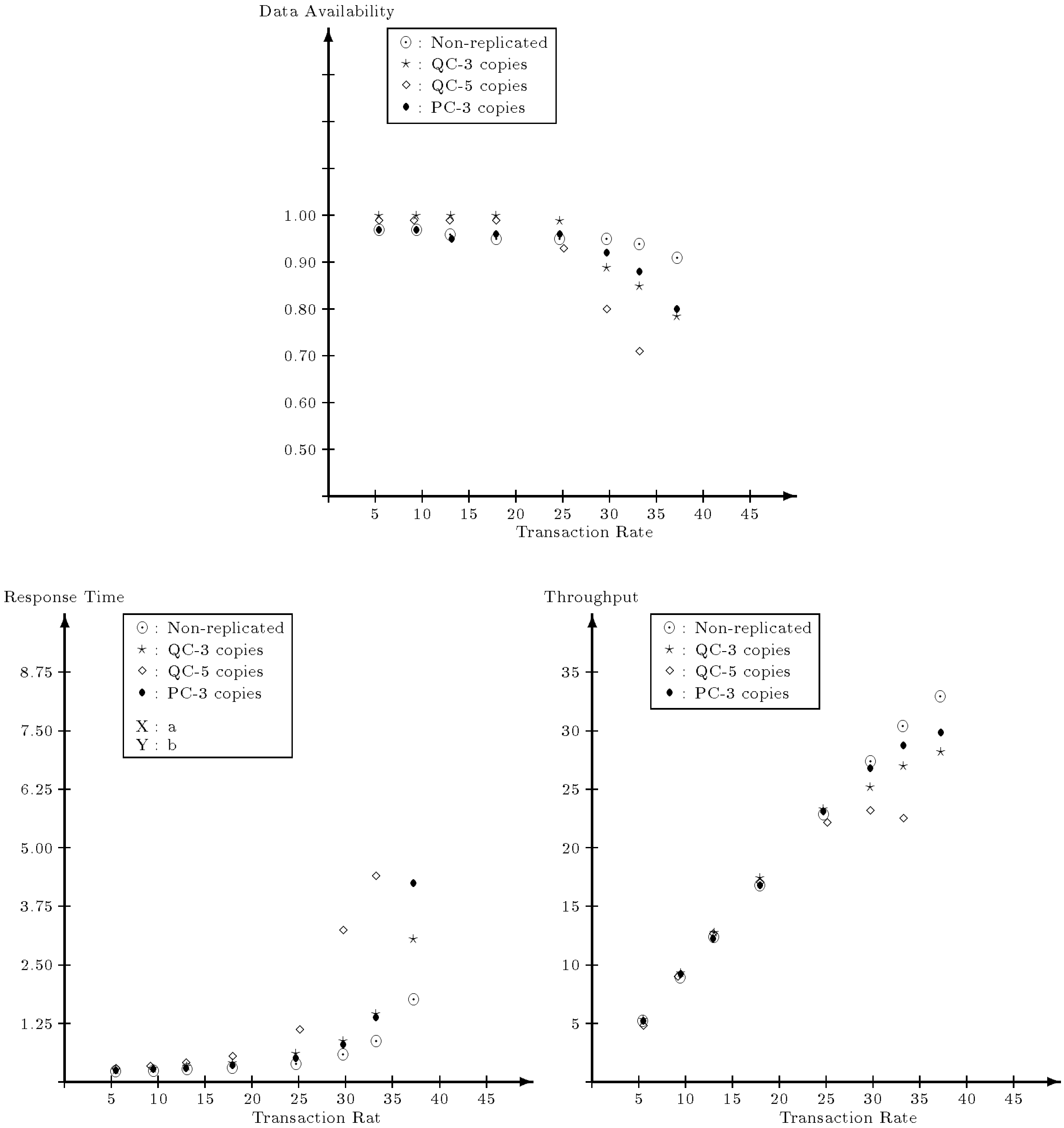

Figure 4: The Effect of Replication, infrequent failure

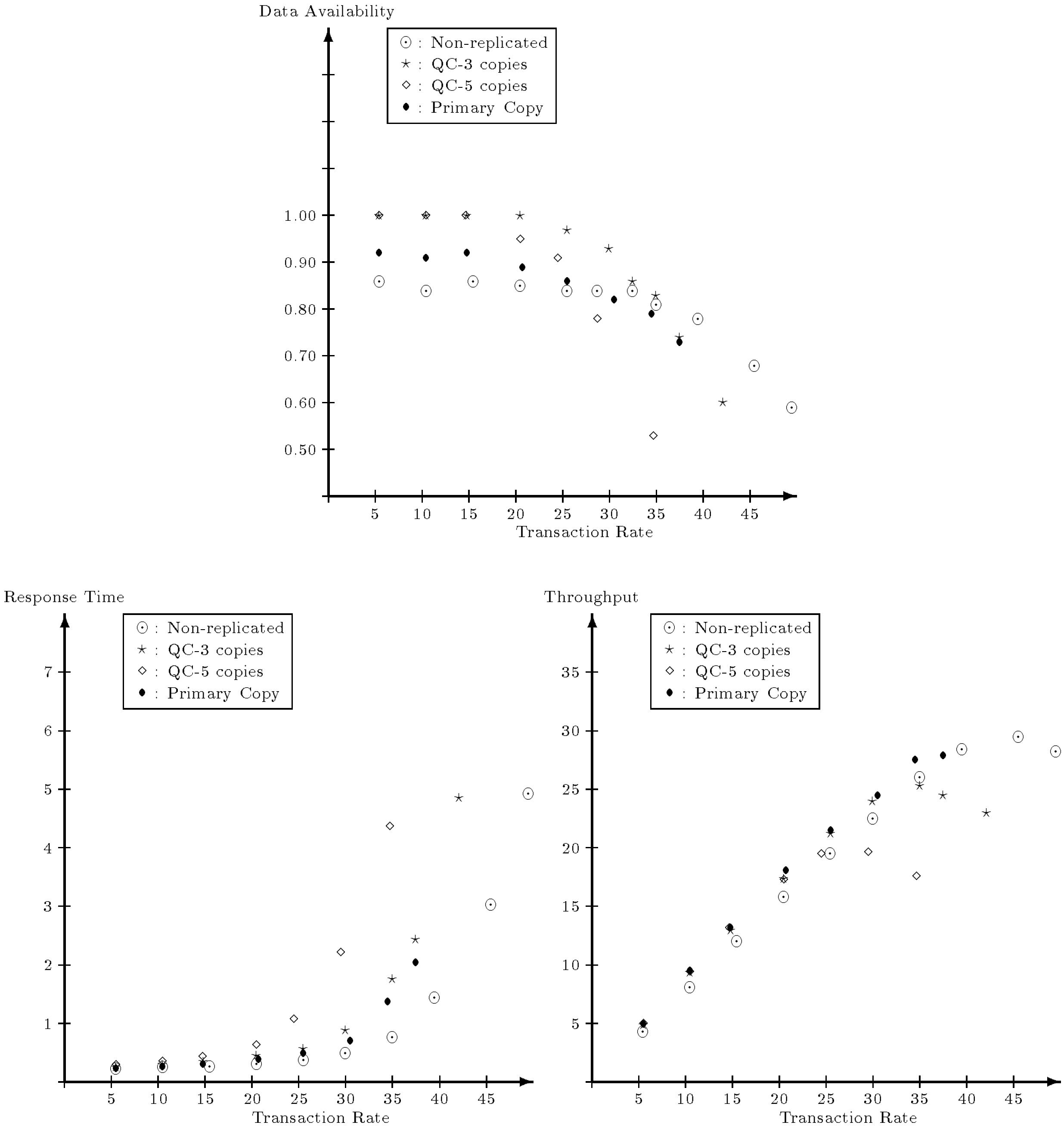

Figure 5: The Effect of Replication, Frequent Failure

under the non-replicated protocol, only very short transactions are likely to commit without timeout (which occurs whenever a failed site does not respond); consequently the response time of a commited transaction also tends to be short.

In the case of PC, the benefit of the replicated data now results in better data availability when compared with the non-replicated case. However, because of the overhead of its view-change process, PC's data availability still compares unfavorably with QC, especially QC-3.

5.4 Long Failure Case

The two cases just presented involve failures occurring at an exaggerated rate and lasting for a fairly short duration. These conditions are biased against the PC protocol in that they exacerbate the overhead incurred by the view-change process, which mitigate the benefit of the data replication. We therefore designed a set of experiments where exactly one long failure is introduced per run: A site fails at the onset of a run and recovers just prior to the run's end (with sufficient time to perform the view change at recovery). In such a setting, the view-change overhead for PC is amortized over the duration of the run, and, as a result, its data availability should now compare favorably with that of the QC protocols. This is indeed verified by the results, which are shown in Figure 6.

6 Discussion

We have employed a simulation testbed of a distributed database system to examine the performance of two replica control protocols, the primary copy and the quorum consensus, in a client-server environment. The results are summarized as follows:

- Until the system's thrashing point is reached, the response time and throughput of a replicated system can be comparable to that of its non-replicated counterpart, during normal operations (i.e, in the absence of failures.)
- Until the system's thrashing point is reached, data availability is enhanced by a replicated system in the presence of failures.
- Under PC, the gain in data availability is offset by the overhead required to maintain consistent replica configurations upon each failure and recovery. The offset is more significant when the failures are frequent and short in duration.
- In a client-server environment, the quorum consensus protocol, which has the virtue of not requiring any special failure and recovery measures, can yield performance characteristics comparable to the primary copy protocol.

We should note that our implementation is intended to reflect the subject protocols in their generic forms. There are many known optimization techniques which can be applied to both PC and QC that are not reflected in our model. As has already been mentioned, the PC protocol implemented on the Harp distributed file system makes use of special hardware to reduce message and logging overhead. Another example is that under QC there are schemes for using bystanders (or witnesses) in lieu of data copies to reduce overhead [27, 28]. Also, under both PC and QC, the copies can be organized in a multi-level hierarchy [20] to reduce the amount of message overhead (however, such a technique is not applicable to our model where the number of copies is small.) The key point is that there are many techniques which can be explored to improve the performance of protocols for replicated databases.

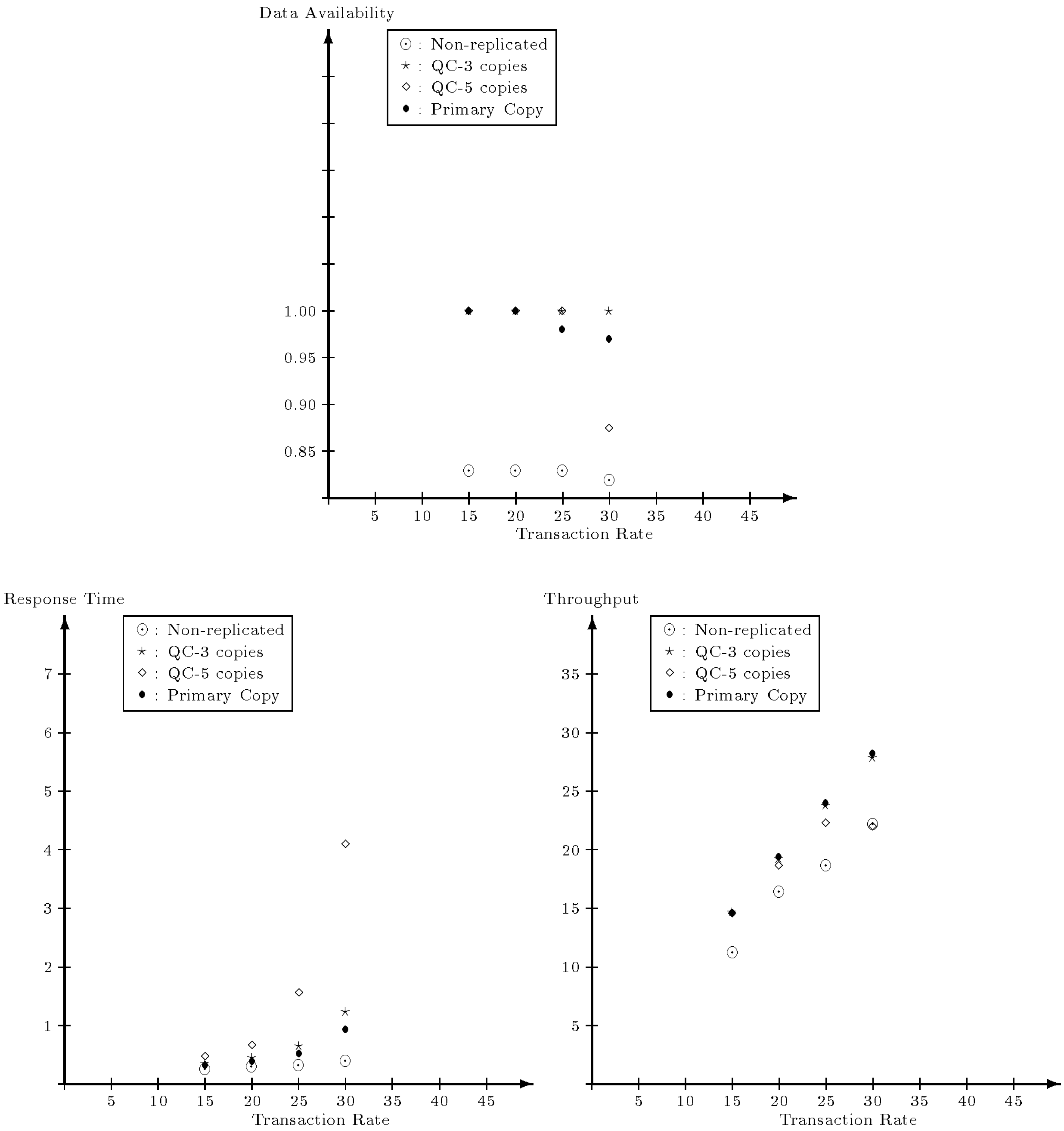

Figure 6: The Effect of Replication, Long Failure

7 Conclusion

In distributed database systems, fault-tolerant data replication is message-intensive. When used indiscriminately, it could have detrimental effects on the performance of a distributed database system. This is a legitimate concern which is perhaps largely responsible for the current lack of acceptance of the data replication technique in the industry, where current applications are often more concerned with high throughput and low response time than with high data availability. Hence the question in the title of this paper. In answer to the question, the findings presented in this paper show that when there is congestion in message processing, data replication does indeed exact a price on the system performance: it results in longer transaction response time and lower throughput, and it also causes a distributed system to reach its thrashing point sooner than its non-replicated counterpart. On the other hand, our findings also indicate that, when there is sufficient resource for message processing, replication has the potential of increasing data availability without significantly sacrificing the conventional performance metrics such as response time and throughput. We conclude that, on a system with sufficient CPU resources, data replication is a viable measure for commercial systems which require a high level of data availability.

References

[1] D. Agrawal and A. Bernstein. A nonblocking quorum consensus protocol for replicated data. *IEEE Transactions on Parallel and Distributed Systems*, 2(2), April 1991.

[2] R. Agrawal, M. Carey, and M. Livny. The Performance of Alternative Strategies for Dealing with Deadlocks in Database Management Systems. *IEEE Transactions on Software Engineering*, SE-13(12):1348–1363, December 1987.

[3] P.A. Alsberg and J.D.Day. A principle for resilient sharing of distributed resources. *Proceedings of the Second International Conference on Software Engineering*, October 1976.

[4] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[5] B. Bhargava, A. Helal, and K. Friesen. Analyzing Availability of Replicated Database Systems. *International Journal in Computer Simulation*, 1(4):345–436, 1991.

[6] B. Bhargava, P. Noll, and D. Sabo. An Experimental Analysis of Replicated Copy Control During Site Failure and Recovery. In *Proceedings of the fourth Internation Conference on Data Engineering*, pages 82–91, February 1988.

[7] M. Carey and M. Livny. Conflict Detection Tradeoffs for Replicated Data. *ACM Transactions on Database Systems*, 16(4):703–746, 1991.

[8] D. Comer. *Internetworking with TCP/IP* . Prentice Hall, 1991.

[9] D. Davcev and W. Burkhard. Consistency and Recovery Control for Replicated Files. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 87–96, December 1985.

[10] A. El Abbadi, D. Skeen, and F. Cristian. A efficient, fault-tolerant protocol for replicated data management. *Proceedings 4th ACM Symposium on the Principle of Database Systems*, pages 240–251, March 1985.

[11] A. El Abbadi and S. Toueg. Maintaining Availability in Partitioned Replicated Databases. *ACM Transactions on Database Systems*, 14(2):264–290, 1989.

[12] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, 19(11):624–633, November 1976.

[13] Domenico Ferrari. *Computer Systems Performance Evaluation.* Prentice Hall, 1978.

[14] D. K. Gifford. Weighted Voting for Replicated Data. *ACM Transactions on Database Systems*, 16(4):150–159, 1979.

[15] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufman, 1993.

[16] M. Herlihy. Dynamic Quorum Adjustment for Partitioned Data. *ACM Transactions on Database Systems*, 12(2):170–194, June 1987.

[17] D.D.E. Long J.-F. Pâris and A. Glockner. A realistic evaluation of consistency algorithms for replicated files. *Proceedings 21st Annual Simulation*, pages 121–130, 1988.

[18] S. Jajodia and D. Mutchler. Dynamic Voting. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 227–238, June 1987.

[19] S. Jajodia and D. Mutchler. Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database. *ACM Transactions on Database Systems*, 15(2):230–280, June 1990.

[20] A. Kumar. Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects. In *Proceedings of the Tenth International Conference on Distributed Computing Systems*, May 1990.

[21] S. Lavenberg. *Computer Performance Modeling Handbook.* Academic Press, 1983.

[22] Averill M. Law and David Kelton. *Simulation Modeling and Analysis.* McGraw Hill, 1991.

[23] M. Liu, A. Agrawal, and A. El Abbadi. A simple and efficient implementation of the quorum consensus protocol. *In Submission*, 1994.

[24] M. L. Liu. *The Design of Distributed Database Systems in the Presence of Failures.* PhD thesis, The University of California, Santa Barbara, 1994. In preparation.

[25] B. Oki and B. Liskov. Viewstamped Replicataion: A New Primary Copy Method to Support Highly-Available Distributed Systems. *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, pages 8–17, 1988.

[26] M. T. Ozsu and P. Valduriez. Distributed Database Systems: Where Are We Now? *Computer*, 24(8):68–78, August 1991.

[27] J. F. Pâris. Voting with Witnesses: A Consistency Scheme for Replicated Files. In *Proceedings of the Sixth International Conference on Distributed Computing Systems*, pages 606–612, June 1986.

[28] J. F. Pâris. Voting with Bystanders. In *Proceedings of the Ninth International Conference on Distributed Computing Systems*, pages 394–401, June 1989.

[29] J. F. Pâris and D. E. Long. Efficient Dynamic Voting Algorithms. In *Proceedings of the Fourth IEEE International Conference on Data Engineering*, pages 268–275, February 1988.

[30] C. Pu, J. D. Noe, and A. Proudfoot. Regeneration of Replicated Objects: A Technique and its Eden Implementation. *IEEE Transactions on Software Engineering*, 14(7):936–945, July 1988.

[31] J. Seguin, G. Sergeant, and P. Wilms. A majority consensus algorithm for the consistency of duplicated and distributed information. *Proc. IEEE Int. Conf. Distributed Computing Systems*, pages 617–624, 1979.

[32] M. Stonebraker. Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRESS. *IEEE Transactions on Software Engineering*, pages 188–194, May 1979.

[33] M. StoneBraker and E. Neuhold. A distributed data base version of ingres. *Proceedings 2nd Berkeley Workshop on Distributed Data Bases and Computer Networks*, May 1977.

[34] R. H. Thomas. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

[35] Z. Tong and R. Y. Kain. Vote Assignments in Weighted Voting Mechanisms. In *Proceedings of the Seventh Symposium on Reliable Distributed Systems*, pages 138–143, October 1988.

[36] R. van Rennesse and A. Tanenbaum. Voting with Ghosts. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*, pages 456–462, June 1988.