Adaptability Experiments in the RAID Distributed Database System

Bharat Bhargava Karl Friesen
Abdelsalam Hela John Riedlt

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
(317) 494-6013

Abstract
Adaptable systems can improve reliability and performance by allowing dynamic reconfiguration. We are conducting a series of experiments on the RAID distributed database system to study the performance and reliability implications of providing static and dynamic adaptability. Our studies of the cost of our adaptable implementation were conducted in the context of the concurrency controller and the replication controller. We show that adaptable implementations can be provided at costs comparable to those of special purpose implementations. The experimentation with dynamic adaptability focuses on concurrency control. We show that dynamic adaptability can result in performance benefits and that system reconfiguration can be accomplished dynamically with less cost than stopping the system, performing reconfiguration, and then restarting the system. Our examination of the costs of providing greater data availability studies the replication control and atomicity control subsystems of RAID. We demonstrate the cost associated with increasing availability in an adaptable scheme of replication control and commit protocols.

1 Introduction
Adaptability and reconfigurability are needed to deal with the changing performance and reliability requirements of a distributed system. An adaptable system can meet a variety of application needs in the short term, and can take advantage of advances in technology over the years. Our previous work allows an entire system to convert from one concurrency control method to another without stopping or aborting transactions [BR89a]. Adaptability in reconfiguration algorithms has also been studied. Dynamic quorum methods change the replication control method during failures to increase availability [Her87, BB90, BGMS86]. Several adaptable distributed systems have been built [MKSSS, BR89b, BHJt86]. Projects are underway to increase the adaptability of operating systems [BM84, Che88, BM89].

There is a need for further work on the performance costs of an adaptable implementation, the costs of dynamic adaptation, and the problem of deciding when to perform system adaptation [BR89a]. We have conducted experiments that investigate adaptable recovery when failures and network partitions occur [BB90]. We are conducting scientific experiments on the RAID distributed database system that focus on three issues: the performance costs of providing an adaptable implementation, the specific costs of dynamic adaptability, and the costs attributable to increased data availability. Push [BMR89] is another project that facilitates experimental work in adaptability by allowing users to extend kernel-level services to enhance database performance.

The remainder of this section is devoted to a brief description of RAID. RAID [BR89b] is being developed on SUN workstations under the UNIX operating system. RAID has proven useful in supporting experiments in communication [BMR87, BMR], adaptability [BB90], and transaction processing [BR89b]. The RAID group has
2 Experimental Infrastructure

The Raid laboratory has five Sun 3/50s, and four Sun SparcStation-1s, all with local disks and connected by a 10Mb/s Ethernet. Measurements are facilitated by microsecond resolution timers that were obtained from Zytec Corporation. In the following subsections, we discuss the adaptability features of RAID. We also describe the benchmarks for our experiments and the action driver simulator which parameterizes and applies the benchmark to the RAID system. We also discuss the stability of open and closed experiments. Finally, our experimental procedure is presented.

1In the rest of the paper RAID will be used to mean RAID-V2. RAID-V2 will be used only for emphasis or to improve clarity.

2.1 Adaptability Features in RAID

Currently, three RAID servers have built-in adaptability features — the concurrency controller (CC), the replication controller (RC), and the atomicity controller (AC). Each of these servers implements a number of algorithms and has the mechanisms necessary to convert from one algorithm to another.

The CC implements five algorithms for concurrency control: timestamp ordering (T/O), two-phase locking (2PL), generic timestamp ordering (gen-T/O), generic locking (gen-2PL) and generic optimistic (gen-OPT). The first two algorithms are implemented using specialized data structures, while the last three use the same general data structures. In the case of T/O and gen-T/O, the implementations enforce the same concurrency control policy, but one uses a generic data structure specifically designed for adaptability, while the other uses a data structure designed specifically for T/O.

The RC implements an adaptable version of the quorum consensus (QC) algorithm [Gil79], where the quorum assignments are determined by a quorum-parameters relation. Quorum assignments may be changed by updating this relation. Many of the standard replication control methods can be expressed using this mechanism. The quorum parameters may be set to enforce the read-one-write-all policy (QC-ROWA), the read-same-as-write policy (QC-RSW), or read-all-write-one policy (QC-RAWO), to name just a few of the possibilities. QC-ROWA requires that any read quorum must contain at least one site, and any write quorum must involve all sites. Similarly, QC-RAWO requires that any read quorum must be able to access all sites, but only one site is required to perform a write operation. QC-RSW requires both read and write quorums to be comprised of a majority of sites. Since QC-RSW does not require all sites to be operational in order to form its quorums, it provides a greater degree of availability than QC-ROWA or QC-RAWO. The RC also implements a version of read-one-write-all (ROWA) that does not employ the quorum mechanism. This allowed us to test the performance cost of our quorum implementation.

The AC implements centralized two-phase commit (2PC) and centralized three-phase commit (3PC). Transactions in the AC are independent of each other, so the selection of a commit protocol can be performed on a per-transaction basis. In practice, this selection is done by the RC, which may elect to use the default AC protocol.

Adaptability features in RAID make it possible to test different algorithms and implementation techniques under the same conditions using the same benchmarks. A single independent variable can be chosen, and can be varied over a range of values while the rest of the system...
remains constant. For instance, many different replication controllers can be tested with the same atomicity controller, concurrency controller, and access manager, and under the same workload.

### 2.2 Benchmark Data

Several benchmarks for database systems exist [BDT83, A+85]. In the DebitCredit (or TP1 or ET1) benchmark [A+85], a transaction reads and writes a single tuple from each of three relations: the teller relation, the branch relation, and the account relation. In addition, a tuple is appended to a special write-only sequential history file describing the transaction. The benchmark requires that the entire transaction be serializable and recoverable.

In order to obtain a greater variety of transaction streams, we extended the DebitCredit benchmark to support changes in transaction length, percent of accesses that are updates, and percent of accesses that are to hot-spot items. Each transaction consists of some number of actions, each of which accesses a random tuple of one of the three relations. The access is either an update of the balance field or a select on the key field. Some percentage of the updates are directed to a hot-spot of the relation, which is the first few tuples of that relation. Each transaction that performs at least one update ends with an insert to the history relation.

We built a transaction generator to generate a stream of random DebitCredit transactions based on the following input parameters:

- **transactions**: number of transactions to generate.
- **branches**: number of tuples in branch relation.
- **tellers**: number of tuples in tellers relation.
- **accounts**: number of tuples in accounts relation.
- **average length**: average number of actions in a transaction. Transaction length is a normal distribution, with standard deviation 1/3 of the mean.
- **update fraction**: fraction of the actions that are updates.
- **hot-spot size**: fraction of the database comprising the hot-spot. Each action is checked to see if it should be a hot-spot action. If so, it accesses tuples number \([0, \ldots, \text{hot-spot size} \times n_{\text{tuples}}]\) for the chosen relation. All relations have the same hot-spot size.
- **hot-spot access fraction**: the chance that an action on a relation will access the hot-spot of that relation.

The hot-spot is over a fixed number of tuples across three relations. Since RAID does tuple-level locking, this
is the same as having a single hot-spot in one of the relations.

2.3 Action Driver Simulator

The AD simulator is used in place of the RAID AD to provide an easily controllable workload. It interprets commands written in a benchmark language and generates transactions according to these specifications. All of the transactions that can be generated by the Debit Credit benchmark can be generated by the AD simulator.

The AD simulator accepts a parameter $\beta$ that specifies the inter-arrival rate of the transactions. $\beta$ is used as the mean for an exponential random variable. When the arrival of a transaction is indicated, the AD generates read and write actions according to the benchmark commands. This information is communicated to the rest of the system via the RC. Each transaction is timed while it is running. If it runs out of time the AD assumes that a deadlock occurred, or that a message has been lost and restarts the transaction. The timeout interval is a constant number of milliseconds per action, chosen to maximize system throughput.

RAID supports dynamic adaptability by including a special fully replicated control relation in each database. Control transactions are processed like normal transactions until they are committed. At this point each server checks to see if the transaction is a control transaction for that server. If so, the server interprets the update as a dynamic adaptability request. Since control transactions are serialized like other transactions, there is automatic protections against multiple operators introducing an inconsistent state.

2.4 Open versus Closed Experiments

The AD simulator is set up to run two basic types of experiments. In open experiments the transaction inter-arrival gap is varied to control the system load. In closed experiments the multiprogramming level is fixed. When one transaction completes another is started. We found that results of the closed experiments were consistently easier to understand and interpret than the results of the open experiments. At a high degree of concurrency an open system is very unstable especially in the presence of transaction restarts.

In most applications, the successful completion of transactions is required. In such applications, transactions aborted by the transaction manager must be retried until they succeed. In order to model such behavior in our experiments, transactions aborted by the system were restarted by the AD. We ran a series of experiments on the effect of the restart policy on throughput [Rie90]. We found that a random delay that was based a rolling average of transaction processing time, produced a restart policy that was responsive and that maintained system stability.

2.5 Experimental Procedure

All experiments have been run early in the morning, when network activity was low. All of the RAID machines were first rebooted to ensure that the experiments would run on a "clean" system. After the reboot, a file was read to determine parameters for the experiment. For each set of parameters, a new RAID instance was started. Transactions were generated by the Action Driver simulator, and processed by the system. When a RAID instance terminated, each server wrote out statistics for the run. The next set of parameters was then processed. All of the machines involved in these experiments utilized a local disk for the transaction log, as well as for database accesses and updates.

Each experiment involved running 250 transactions on the system. 250 was chosen as a reasonable number that yielded approximately steady-state measurements despite the start-up and tail-off times. There was little difference between running 250 transactions and running 300 transactions.

Unless otherwise specified, the experiments were run with the extended Debit Credit benchmark using the following parameters: mean transaction length of 4, 50% update percent, 80% of the actions access the hot-spot, and a 20% hot-spot size.

Unless otherwise indicated, experiments were run on a baseline system that used a timestamp ordering concurrency controller, a read-one-write-all replication controller, and a two-phase commit protocol. All experiments were "closed" with a fixed degree of multiprogramming. The degree of multiprogramming was set to a small number (3) to minimize serialization conflicts. Aborted transactions were restarted after a delay that was computed using an exponential random distribution with the rolling average of transaction response time as the mean.

In each experiment the workload was provided by a single AD running on one of the sites.

3 Experiments in Adaptability

We describe three experiments on adaptability. The first experiment measures the overhead of an adaptable implementation in the concurrency controller and replication controller subsystems of RAID. The second experiment identifies a set of conditions under which concur-
rency control adaptation could be beneficial. The third experiment measures the cost of replication and atomicity control methods that increase availability. The choice of servers for a given experiment was based on the current infrastructure available in RAID and the potential to obtain meaningful data.

3.1 Experiment I: Cost of Adaptable Implementation

3.1.1 Statement of the Problem

We believe that an adaptable design is more complicated than a non-adaptable design. This experiment measures the difference in performance between the adaptable and the non-adaptable implementations.

3.1.2 Procedure

We conducted this experiment on the CC and the RC subsystems of RAID. Of the five concurrency controllers implemented in the RAID CC, two are implementations of the timestamp order (T/O) policy. These implementations enforce the same concurrency control policy, but one uses a generic data structure specifically designed for adaptability, while the other uses an ad hoc data structure designed specifically for T/O. To compare the cost of using adaptable implementations, we measured the non-adaptable T/O algorithm against the adaptable T/O method varying the size of the database hot-spot. The experiment was run on a single-site database with a small hot-spot to produce a significant degree of conflict.

For replication control, we measured the non-adaptable read-one-write-all method (ROWA) against the adaptable quorum-based implementation of the same method (QC-ROWA). The overhead incurred by the QC-ROWA implementation includes constructing the most-up-to-date value for every read quorum and packing new version numbers for every write quorum, among others. To measure the cost of the adaptable implementation, we measured the response time of both ROWA and QC-ROWA, varying the update percent. A four site system was used for this portion of the experiment.

3.1.3 Data

Figure 2 shows the throughput for the two concurrency controllers as the size of the hot spot increases. Figure 3 shows the response time for the ROWA and the QC-ROWA methods as update percent increases. The 90% confidence intervals for both sets of data were less than 10% of the data values. All data shown is from systems running on Sun 3/50s.
3.1.4 Discussion
To the limits of the experiment there were no discernible differences in performance between the adaptable implementations and the specialized implementation. The reason is that the differences in execution time between the two algorithms are small in comparison to the execution time required to process a transaction. The selected algorithm has greater impact on performance than different implementations and execution speed.

For the replication control data, we observe that the cost of the adaptable method (QC-ROWA) is not significantly different from the cost of the non-adaptable method (ROWA). In the case of concurrency control and replication control, a carefully designed adaptable implementation can perform as well as a non-adaptable implementation.

3.2 Experiment II: Cost and Benefit of Dynamic Adaptability
3.2.1 Statement of the Problem
Dynamic adaptability allows the operator of a system to change from one algorithm to another while the system is running. We examine the cost of adaptation to determine the effectiveness of particular dynamic adaptations.

3.2.2 Procedure
This experiment was performed on the CC server. The item-based generic state described in [BR89a] was used to implement three concurrency controllers: generic 2PL, generic T/O, and generic OPT. Then four conversion routines were written to dynamically convert from generic 2PL to and from each of generic T/O and generic OPT, while preserving correctness. To preserve correctness aborts are sometimes necessary. A special benchmark was set up to test these conversion routines. This benchmark first ran a control transaction to convert to the initial concurrency controller, then ran 50 transactions to get the system to steady state, and then ran another control transaction to convert to the final concurrency controller. Finally 20 more transactions were run to ensure that the second control transaction ran under normal conditions. The multiprogramming level was set to 20 for these experiments to increase the number of transactions to be checked for abort. The number of aborts required during adaptation (called convert abort) are reported to represent the cost of dynamic adaptation.

Measuring only aborts excludes the computation cost of the actual conversion from one method to another. In all methods except 2PL to OPT (which has no conversion cost) this cost is proportional to the number of elements of the read sets of active transactions. In RAID this cost is a small fraction of transaction processing time.

The benefit, on the other hand, is sustained over time, and is in the form of increased throughput from running a better algorithm for the current transaction mix. A measure of the net gain for dynamic adaptability is the amount of time required to make up for the cost, assuming the transaction characteristics remain the same. Thus, we propose that the expected break-even time be defined by

\[
t = \frac{\text{aborts during conversion}}{\text{abort rate}_{\text{old}} - \text{abort rate}_{\text{new}}}
\]

The numerator in this expression expresses the cost of the conversion in aborts. The denominator has units aborts/second, and expresses the benefit of running the new algorithm. The units of the result are seconds, and it expresses the amount of time the system must run with the new method and the same transaction processing conditions to recover the cost of conversion. An alternative expression for net cost of abort is to convert the cost of conversion to throughput, and express the benefit in terms of the increased throughput of the system after conversion:

\[
t = \frac{\text{lost throughput during conversion}}{\text{throughput}_{\text{new}} - \text{throughput}_{\text{old}}}
\]

Here throughput is expressed in transactions per second, and lost throughput during conversion is computed by estimating the throughput cost of the aborts. One such estimate is to subtract the number of whole transaction equivalents that were aborted. For instance, one transaction \(\frac{1}{2}\) complete and one transaction \(\frac{1}{3}\) complete would be combined to make one transaction equivalent.

In general, the former measure is easier to compute since the number of aborts during conversion is readily available, but the latter measure is a more accurate measure of the actual estimate of the conversion cost, especially if a good estimate of the number of whole transaction equivalents is available.

We measured the number of aborts required for each type of conversion over a range of benchmarks under a high degree of multiprogramming. We measure the throughput of the old and new methods for each type of conversion. We compute the net cost of the conversion, in seconds.

3.2.3 Data
The number of aborts required for dynamic conversion under a multi-programming level of 20, for a range of hot spot sizes are shown in figure 4. The relative confidence intervals for this data are very large (in some cases as great as 100% of the data values), so the data should not
be interpreted as good indicators of the number of aborts needed for conversion for each hot-spot size. However, in no case more than three aborts were needed during conversion, and for every hot-spot size the average number of aborts was no greater than two. Note that converting from 2PL to OPT and 2PL to T/O never requires aborting a transaction. The 2PL to OPT case never requires an abort because OPT uses the same rules as 2PL, but applies them at the end of the transaction. The 2PL to T/O case never requires an abort in theory because committed transactions can be implicitly renumbered to have earlier timestamps than all uncommitted transactions, thus avoiding all out-of-order timestamp conflicts.

Figure 5 shows similar data for a two-site RAID instance, with transaction length as the independent variable. All the concurrency controllers were converted to the new method in a single control transaction. Again the data is not consistent, but the maximum number of aborts was eight, and the average is never much more than five aborts.

Figure 6 shows the net cost of dynamic conversion from generic 2PL to generic OPT, assuming that two transactions are aborted during conversion and that the multiprogramming level after conversion is 10. We use the throughput-based method of computing net conversion cost, and assume that each aborted transaction was halfway completed.
3.2.4 Discussion

In order to invoke dynamic adaptation, a system manager must have some measure of the benefit of adaptability versus the cost. In the case of the generic state adaptability used in the previous experiment the cost is instantaneous, and is measured in number of aborted transactions. A system manager reading figure 6 would look up the current hot-spot size and read the number of seconds the system would have to stay at this hot-spot size before the cost of converting to OPT would be regained. In this case he/she would be very likely to perform the conversion, since less than one second is required to make up the lost transactions.

In all cases the cost of conversion was relatively low in number of aborts. This is to be expected because only active transactions are candidates for abort using the generic state conversion method, so even at a degree of multiprogramming of 20 there are at most 20 transactions that could be aborted. Note that the number of aborts required for generic state adaptability is exactly the same as the number that would be required for converting state adaptability. Converting state adaptability would have a higher computation cost, but the difference is unlikely to be as high as even one transaction execution time. Estimating one extra lost transaction of cost during conversion still yields a net cost of less than two seconds for conversion.

Dynamic adaptability of concurrency control is inexpensive. Even under a heavy load the cost can be amortized in less than one second if the new algorithm is a significant improvement over the old algorithm.

3.3 Experiment III: Cost Attributable to Increased Availability

3.3.1 Statement of the Problem

Many replication control and commit protocols provide increased availability at the cost of performance. This experiment attempts to quantify the cost involved in adapting to such protocols. For the replication control, we compare the quorum-based read-one-write-all method (ROWA) against the quorum-based read-same-as-write method (QC-RSW). Unlike QC-ROWA, the QC-RSW method requires only a majority of sites to be operational in order to form its read or write quorums. Such tolerance to multiple failures increases system availability at the cost of additional reads, especially for low update percentages. The QC-RSW method is explained in section 2.1.

For the atomicity control, we examine the performance of the two-phase commit (2PC) and the three-phase commit (3PC) protocols.

Figure 7 illustrates the difference in response time between QC-ROWA and QC-RSW that provides higher availability.

Figure 8 shows the throughput for 2PC and 3PC on a five-site RAID system, running on Sun 3/50s. Figure 9 shows the average commit time for the series of experiments. The 90% confidence intervals for both figures are
less than 10% of the data values.

3.3.4 Discussion

The QC-ROWA replication method consistently performed better than the QC-RSW method. In the cases where the update percent is low, the differences between the two methods are not as significant. In another experiment (that we could not report in this paper for space limitation), we varied the degree of replication from 1 to 9 copies. Similar results were obtained regarding the relative cost of QC-ROWA and QC-RSW, in terms of response time.

The difference in throughput for the AC was much less pronounced. Although 3PC consistently required more time than 2PC to complete, the impact of this difference on system throughput is not statistically significant. The difference in commit time, reflecting the AC cost alone, is more pronounced, with 3PC requiring 30% to 50% more time than 2PC. The lack of effect on throughput of the commit protocol reflects the relatively low contribution of distributed commitment to the total transaction processing time. Systems that have lower transaction processing costs than RAID will see a more significant effect on the system throughput when running the 3PC protocol. A similar experiment that used the transaction length instead of the update fraction as the independent variable yielded equivalent results.

In summary, there is some advantage that could be accrued by a system that can utilize more efficient methods (such as ROWA or 2PC) during normal transaction processing and can still provide increased availability when it is needed. Systems with relatively heavy-weight transactions, like RAID, can use 3PC all the time for increased availability at little cost. Systems with lighter-weight transactions may not be able to afford the cost of running 3PC all of the time. Such systems could benefit from an adaptable commit protocol that can change between 2PC and 3PC as needed to provide better availability.

4 Conclusion

These experiments show that for certain types of transaction systems adaptability can be a useful tool for improving performance and availability. Furthermore these experiments establish a method for determining a spectrum of values of the five independent variables (transaction length, percent updates, inter-arrival gap, hot-spot, and transaction granularity) for which adaptability is beneficial in a distributed transaction processing system.

Experiment 1 shows that the overhead of a careful adaptable design can be kept low, with respect to the
overhead of a non-adaptable design.

Experiment II shows that dynamic switching among algorithms while processing transactions can be done efficiently.

Experiment III shows that employing algorithms that can increase availability can be expensive. The experiment suggests that adaptable implementations that can change between high-performance methods and high-availability methods can serve to increase system availability at a relatively low cost in performance.

An extension of these results would be to automate the decision to adapt to a new algorithm through the use of an expert system. The expert system could use the results of experiments like these to determine the approximate costs and benefits of dynamic adaptation in a given situation.

References


