

*Norwegian University of Science and
Technology
Technical Report IDI-TR-06/2007*

**Performance of Main Memory Commit
Protocols**

Heine Kolltveit and Svein-Olaf Hvasshovd
Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway

{heine.kolltveit,svein-olaf.hvasshovd}@idi.ntnu.no

Abstract

Distributed transaction systems require an atomic commitment protocol to preserve ACID properties. The performance of such protocols must be properly evaluated to make system developers able to make an educated choice. In this report, simulation of throughput and response time of atomic commitment protocols for main memory systems are presented. The simulations are verified by statistical analysis. Also, various performance enhancers for main memory commit protocols are presented and evaluated. The simulation results show that the performance can be significantly boosted by using optimizations and protocols especially designed for main memory systems.

1 Introduction

Declining main memory (MM) prices and increasing storage capacity have contributed to the advent of MM databases and applications. In contrary to traditional disk resident databases, main memory databases (MMDB) avoids slow disk accesses by *permanently* storing all data in main memory [1]. This speeds up data accesses, since accesses to the main memory is much faster than accesses to a disk [2, 1].

MMDBs can be classified according to the *log policy* and their *persistence policy*. The log policy determines if the log is saved to disk, or only kept in main memory. The persistence policy determines if the log is persistently stored or not. When logging to disk, the log is persistently stored if *synchronous logging* is used. If not, the log cannot be guaranteed to be recoverable. In *pure* main memory systems, neither the data or the log resides at the disk. The log and data can be made persistent by *replicating* it between processes. This report is in the context of persistent pure main memory databases.

To replicate data between processes, messages are sent between them. Rather than the long disk latency for disk-based systems, a shorter network latency is introduced. Thus, the performance of a MM system has the possibility of outperforming a disk-based solution. Also, since there exist multiple copies of the data, a crash failure [3] will only bring down one of the copies. Thus, high data availability is achieved, giving better performance under failures.

To enable MM systems to exploit the potential advantages, tailored protocols are needed. A transactional system, such as a database, require an *Atomic Commitment Protocol* (ACP) to ensure ACID properties [4]. Many ACPs have been designed [5, 6], but although performance evaluations by simulating commit protocols have been done [7, 8, 9, 10, 11, 12, 13, 14, 15, 6], none are targeted at pure MM protocols.

This report is motivated by the lack of simulation and statistical analysis for MM commit protocols. Without a performance evaluation, the protocols effectiveness have not been sufficiently verified. Since there was no large computing cluster available for testing purposes, simulation was chosen as the method of evaluation. The results of such an evaluation can be favorably used in a shared-nothing fault-tolerant DBMS, e.g., ClustRa [16].

The contribution of this report is analytically verified simulation results indicating the relative performance of MM commit protocols. Some existing and new optimizations are considered as well. Also, a framework for analysing the performance impact for various MM commit protocols is outlined. These results can be used by system developers to improve the performance of MM applications.

This report only simulates failure-free execution where all transactions commits. Since this is the dominant failure-free execution path, it is where the potential for performance gains are the greatest.

The rest of this report is organized as follows. Section 2 gives an overview of related work. Section 3 presents the commit protocols and optimizations discussed in this report. Section 4 outlines the simulation model and parameters used for the simulations in Section 5. The simulations are compared to a statistical analysis in Section 6, while Section 7 gives some concluding remarks.

2 Related Work

Performance evaluations are presented alongside most proposed commit protocols. For most of them, this involves nothing more than counting messages and log writes required to terminate a transaction, as initially done for 2PC in [17].

Simulation results comparing different types of commit processing exist for some cases, most of them assume logging to disk. The impact of communication and site failures for 2PC, optimistic 2PC and a version which resends requests before giving up are presented by Boutros and Desai [7]. Also, Liu, Agrawal and Abbadì [18] simulates 2PC, presumed commit, presumed abort and early prepare in the presence of site failures.

Simulations comparing 2PC, 3PC, presumed abort, presumed commit and an optimistic protocol as well as two baseline protocols, one simulating a completely centralized system and the other distributed transaction processing and centralized commit processing, have been performed [10]. These simulations consider the effects of resource and data contention, the multiprogramming level, slow and fast network interfaces, the degree of distribution, and aborts. Simulations of the same protocols in hard real-time environments, where a transaction is aborted if it is not terminated within a given timeframe, have also been done [12].

In a study presented by Samaras, Georg and Chrysanthis [13], simulation results show that restructuring of the commit tree may yield better response time.

A simulation of commit protocols in gigabit networks [8] compares the performance of the three types of standard two-phase commit (basic, presumed abort and presumed commit [19]) and two one-phased protocols (Coordinator Log [20] and Implicit Yes-Vote (IYV) [21]). The study shows that the one-phased protocols, with IYV at the top, are better, when applicable, for high speed networks.

A simulation study for main-memory commit protocols exist [14], which is further refined in [15, 6]. However, this approach and the simulation assumes logging to disk, thus the commit processing is severely degraded compared to a pure main-memory approach.

Many statistical queuing analyses of transactional systems have been performed, and several surveys of them have been conducted [22, 23, 24]. They are mostly focused on other aspects of transaction processing, mainly concurrency control, and the mathematical analysis of different types of commit processing have been overlooked.

3 Commit Protocols

This section first presents some commit protocols for main-memory primary-backup systems. They are introduced by outlining the standard commit protocols that have influenced them. Second, some specific and some general performance enhancers for commit processing are introduced.

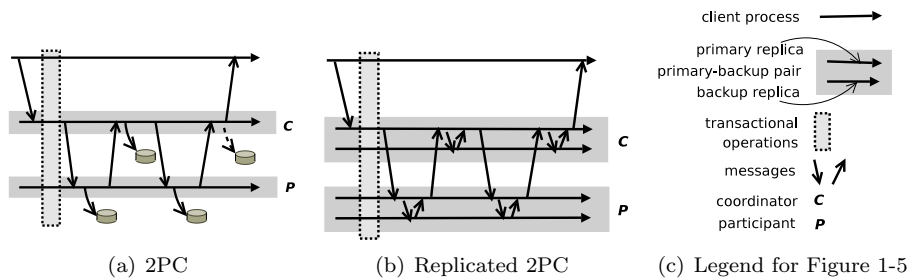


Figure 1: Failure free execution of 2PC and R2PC

3.1 2PC

The Two-Phase Commit Protocol (2PC) [17, 25] consists of two phases: The *voting phase* and the *decision phase*. Figure 1(a) illustrates the protocol for one participant. During the voting phase the votes are collected from the transaction participants by the transaction coordinator by sending out a **Prepare** message. The participants make their vote *durable* by forcing to stable storage [26]. Then, each of them replies with a **Vote** message. The coordinator makes a decision based on the votes. A missing or negative vote is interpreted as a veto, and it decides to **ABORT** the transaction. Otherwise, the decision is **COMMIT**. The decision is then forced to stable storage.

In the decision phase, the decision is propagated to the participants, which force the decision to stable storage and reply **Committed** to the coordinator and forget the transaction. After receiving **Committed** from all participants, the coordinator writes a committed log record and forgets about the transaction. This record does not need to be forced, as indicated by the dashed arrow to the disk in Figure 1(a).

3.2 R2PC

The *Replicated Two-Phase Commit Protocol* (R2PC) is a simple modification of 2PC to a main-memory primary-backup environment. In such an environment the forced and non-forced disk writes can be replaced by, respectively, synchronous (blocking) and asynchronous (non-blocking) logging to the backup node. Figure 1(b) illustrates this. The small arrows between each primary-backup pair is the logging. The rest of the protocol remains unchanged.

3.3 C2PC

The *Circular Two-Phase Commit Protocol* (C2PC) [5] is a protocol designed especially for main-memory primary-backup systems. As the name indicates, it has the same two phases as 2PC and R2PC. However, as shown in Figure 2 the commit processing is executed in a circular fashion to reduce the number of messages. The primary coordinator initiates the commit processing by sending **Prepare** to all primary participants. Each primary participant piggybacks its vote on a **Prepare** message to its corresponding backup participant. Each of the backup participants piggybacks its vote on a **Prepare** message to the backup coordinator, which collects the votes and makes a decision.

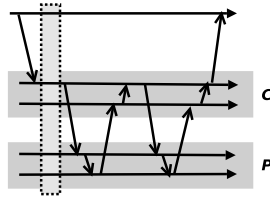


Figure 2: Failure free execution of C2PC

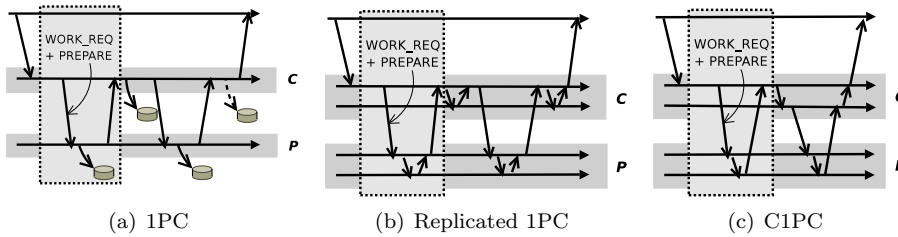


Figure 3: One-phased commit protocols

In the decision phase, the decision is sent to the primary coordinator and subsequently propagated to the primary and backup participants in the same fashion as the `Prepare` message. The backup participants acknowledges the decision by sending `Commit` to the backup coordinator. After the acknowledgements, the participants can forget about the transaction. Finally, the primary coordinator receives `Committed` from the backup coordinator and the transaction is completed.

A more detailed explanation and proof of correctness for C2PC is given in [5].

3.4 One-Phased Protocols

One-phased protocols skip the voting phase by making assumptions about the participants which can cause severe restrictions on the execution of transactions [27]. The oldest is the *Unsolicited Vote protocol* (UV) outlined by Stonebraker [28] where `Prepare` is piggybacked on `DoWork`. Thus, the voting phase is included in the transaction processing as shown in Figure 3(a). Again, a simple version for main-memory primary-backup systems is developed. In *Replicated One-Phase Commit Protocol*, R1PC, the disk writes has been replaced by sending messages to the backups as shown in Figure 3(b).

The *Circular One-Phase Commit Protocol* (C1PC) [5] is a one-phase version of C2PC, based on UV. The backup participants send their votes to the primary coordinator instead of the backup coordinator as in C2PC. This is done since the primary coordinator may need to do additional work after the results have been received and before preparing the transaction locally. Thus, the primary coordinator collects the votes and makes a decision which is sent to the backup coordinator. It is responsible for distributing the decision to the primary participants. From there on, C1PC works as C2PC.

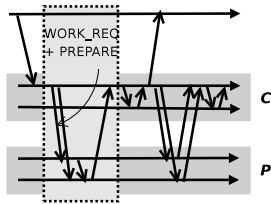


Figure 4: Failure free execution of Clustra

3.4.1 Clustra Commit Protocol

The commit protocol used in Clustra [16], uses a novel one-phased main-memory commit protocol. Figure 4 illustrates the execution. The **primary coordinator** sends out the work requests and piggybacks the prepare message to all **primary participants**. For each **backup participant** it sends out the number of expected log records to receive from the corresponding **primary participant**. When the backup has received the log records, the vote is returned to the **primary coordinator**. The **primary coordinator** persistently stores the decision and identifiers of the participants to the **backup coordinator** and give an *early answer* to the client. Then, all participants receives a **Commit**, and replies with **Committed**. The successful completion of the commit processing is then persistently stored at the **backup coordinator**, before the transaction is completed and forgotten.

In this report, to keep with the syntax of the other optimizations, we call the protocol without the early answer optimization, CCP. The protocol as presented in [16] includes, however, the early answer optimization and is labeled CCP-EA from here on.

CCP normally piggybacks acknowledgement messages on other messages, therefore reducing overhead. The other protocols can employ the same technique. However, this report does not consider the effects of piggybacking. Thus, CCP is also used without piggybacking to ensure a fair comparison.

3.5 Performance Enhancers

Transactions are often sent to a dedicated transaction coordinator node, which coordinates the transaction execution and commit processing. This can cause a bottleneck. A solution is to balance the load and allow all nodes to be transaction coordinators. However, if the transaction coordinator is not a participant of the transaction, this causes unnecessary overhead.

This rest of this section first outlines the early answer optimization used by Clustra and how it can be used on other protocols. Then, it presents some ways to coordinate transaction processing in such a way that the overhead from sending a transaction to a node which is not a participant is limited.

3.5.1 Early Answer

A transaction coordinator does not need to wait until the end of the transaction before it replies to the client of the transaction. It only needs to wait until the decision to commit or abort is persistently stored. This is called *Early Answer* [16], EA. Figure 5 illustrates the optimization. For 2PC, the reply can be given once the decision is saved to disk (see Figure 5(a)) and for C2PC and C1PC the

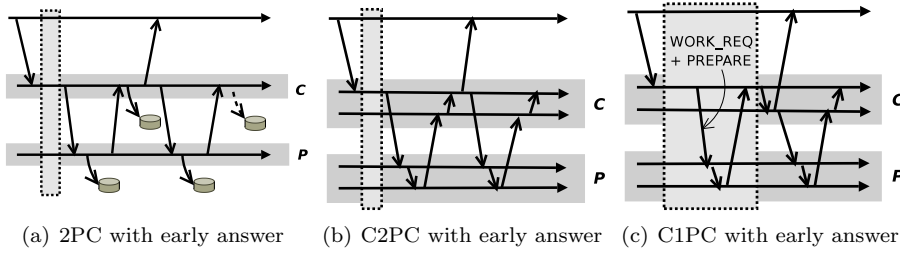


Figure 5: Early answer

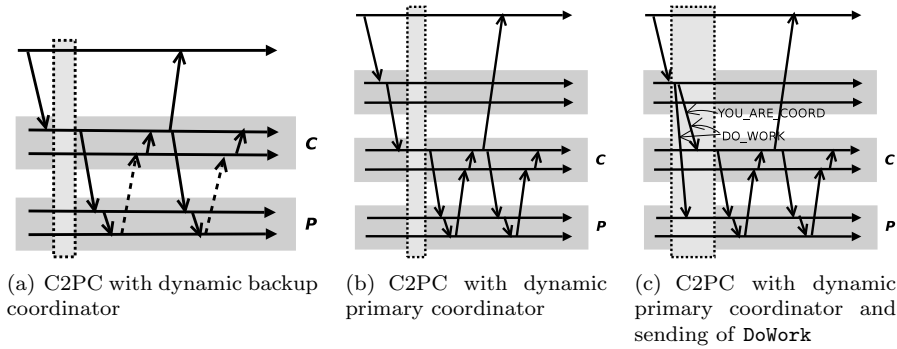


Figure 6: Dynamic roles

reply can be sent as soon as both the primary and backup coordinators know the outcome of the transaction (see Figure 5(b) and 5(c)). The result is that the time to execute the decision phase of the commit processing is not included in the response time. Thus, the time spent doing commit processing is reduced as seen by clients, at no extra costs.

3.5.2 Dynamic Backup Coordinator

One way to reduce the overhead is to make one of the backup participants, become the backup coordinator of the transaction. This optimization is called the *Dynamic Backup Coordinator*, DBC. This will reduce the overhead for C2PC and C1PC, but not for the general commit protocol. DBC is illustrated in Figure 6(a). The dotted arrows are the saved commit processing messages between the chosen backup participant and the backup coordinator. Also, the combined backup coordinator and backup participant prepares and commits as one entity and not as two separated ones. This saves two messages, one prepare task and one commit task. In the figure and during the simulations, DBC are executed with EA.

3.5.3 Dynamic Primary Coordinator

Another optimization is to forward the coordination of each transaction to a primary participant. Thus, one of the primary participant is guaranteed to be the primary coordinator and the corresponding backup participant is the backup coordinator. This optimization is called the *Dynamic Primary Coordinator*, DPC. The forwarding of `BeginTxn` is shown in Figure 6(b). The performance

gain is a complete round of commit processing for one node, minus the extra overhead of forwarding the transaction.

DPC is applicable to all commit protocols described in Section 3. In Figure 6(b) and during the simulations DPC use EA.

3.5.4 Dynamic Primary Coordinator using Piggybacking

DPC can be further improved. Say, a node which is not a participant of the transaction receives a `BeginTxn`. Instead of just forwarding it to a participant it can send `DoWork` to all participants. To one of the participants it can set a `YouAreCoordinator` flag, and to the others it can set a `CoordinatorIs` variable to the address of the new coordinator. This approach is called the *Dynamic Primary Coordinator using Piggybacking*, DPC-P, and is shown in Figure 6(c). Thus, the extra overhead of forwarding the transaction is removed compared to the dynamic primary coordinator approach. This optimization will not add complexity to the failure semantics as all participants know the identities of the chosen primary and backup coordinator. In the figure and during the simulations DPC-P are executed with EA.

4 The Simulation Model

To be able to evaluate the performance of various main memory commit protocols, a realistic simulator of a distributed transaction processing system have been developed. It has been implemented using Desmo-J, a framework for discrete-event modelling and simulation [29].

The distributed transaction processing (TP) system considered in this report is composed of N nodes connected through a communication network, as shown in Figure 7(a). Each node has a *transaction manager* (TM) which is responsible for orchestrating the correct *local* execution of transactions. The correct *global* execution of transactions is ensured by coordination between TMs.

Each transaction has a *coordinator*, which is responsible for the execution and termination of the transaction. Typically, the coordinator is the *transaction manager* (TM) at the node where the transaction was *initiated*. After the transaction execution is finished it is terminated using an atomic commitment protocol between the coordinator and participants TMs.

The distribution of processing and data is transparent to the clients of the transactions. The coordinator is responsible for issuing subtransactions to the appropriate nodes. Nodes receiving subtransactions are called *participants*. Generally, subtransactions can create further subtransactions, causing a multi-level transaction execution tree [19]. However, this report discusses only trees that are one level deep. Transactions are assumed to be precompiled, and all nodes knows where data are located. Thus, the coordinator can forward subtransactions directly to the correct participant.

We model each node to consist of a *transaction manager* (TM) executing on top of the operating system. The *operating system* (OS) is responsible for receiving and sending messages, while the transaction manager processes the requests. Other applications are assumed to be invoked from the TM. For each node, there are three First-In-First-Out queues: One for incoming messages, one for outgoing messages and one for tasks to be processed by the TM. This is

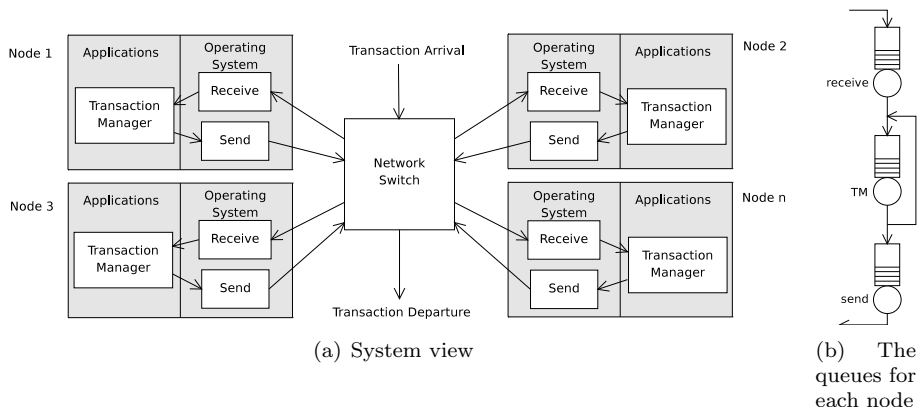


Figure 7: The logical model of the system

shown in Figure 7(b). A processed task, can result in a new local task, which is inserted at the end of the TM-queue, or a remote task, which is inserted into a message and sent by the operating system.

Measurements to provide the input values for the simulations are performed on an AMD Athlon(TM) 64 bits 3800+ processor, running a Linux 2.6 kernel. The length of the transaction operations are derived from measurements performed on a main memory database prototype developed by Løland in the context of derived table creation [30].

An *open* simulation model is used. Process *context switches* between the two processes of the system, OS and TM, are modelled. The OS alternates between sending and receiving messages. Measurements show that these are performed in just $3.5 \mu\text{s}$. The *timeslice* is the maximum time given to each process to execute requests before another process is given time at the cpu. The default timeslice for Linux kernel 2.6 systems is 100 ms and the minimum is 5 ms. To avoid delaying operations too long for this application, a timeslice of 5 ms is chosen.

Each transaction consists of three subtransactions, each of them containing a simple update operation. Thus, during transaction processing each subtransaction makes one visit to its respective participant, and returns. The subtransactions are processed in parallel. The coordinator and participants are dynamically chosen randomly and uniquely for each transaction. None of them are the same, therefore, each of the three subtransactions of a transaction has a unique destination node.

In line with update transactions, the messages sent over the network are assumed to be small, i.e. 200 - 300 bytes. Test experiments on a 100 Mbit/s Ethernet network show that no queuing effects for the network are likely to happen for a distributed main memory database. Also, small messages have near zero transmit time. Thus, the time to send a message from one node to another can be modelled as cpu-time at the receiver and sender. Measurements of CPU usages indicate that it takes twice as long to receive a message, as it does to send it.

Operations are divided into three groups: Long, medium and short. These are made to reflect the various service demands needed by an *average* operation of that kind. Long operations (**DoWork** and **DoWorkAndPrepare**) takes $700 \mu\text{s}$,

Parameter	Value	
Simulation Model	Open	
Context Switch	3.5 μ s	
Timeslice	5 ms	
Send Message	40 μ s	
Receive Message	80 μ s	
Long Operations	700 μ s	DoWork, DoWorkAndPrepare
Medium Operations	150 μ s	BeginTxn, Prepare, Abort, Commit
Short Operations	50 μ s	WorkDone, Vote, Aborted, Committed
Simulation Time	200 s	
Capture Time	160 s [40 - 200]	
Simulation runs	10	
Subtransactions	3	
Transactions	Simple update	
CPU Utilization	Variable {1%, 60%, 80%, 90%, 95%, 97.5%}	
# of simulated nodes	Variable {10, 20, 50}	

Table 1: The input parameters of the simulation

medium operations (**BeginTxn**, **Prepare**, **Abort** and **Commit**) takes 150 μ s and short operations (**WorkDone**, **Vote**, **Aborted** and **Committed**) takes 50 μ s.

As the primary focus of this report is the performance of commit protocols, the database internals are not modelled. A very large number of records are assumed, giving negligible data contention, hence, locking effects can be disregarded.

Each simulation lasted for 200 simulated seconds. The first 40 seconds are disregarded, as we are interested in the *steady-state* system. For each result presented, 10 simulation runs were made.

By running a single transaction, the total service demand of a transaction for each protocol was found. Then, the costs of all but a few of the context switches was subtracted from the total and the answer was used as a divisor. The result is the maximum throughput for each of the protocols per node. To create a graph, the utilization of the system were varied from 1 - 97.5% to see the impact on the system performance. Also, the number of nodes was varied.

Table 1 summarizes the simulation parameters.

5 Simulation Results

Using the system model described in Section 4, simulation experiments were performed. However, to be able to interpret the results, three more detailed experiments were needed.

First, we show that the scheduling of processes on the cpu affects the performance. Second, the impacts of varying the number of participants and the number of simulated nodes are examined. Third, the results of varying the number of nodes and utilization during simulation of various commit protocols

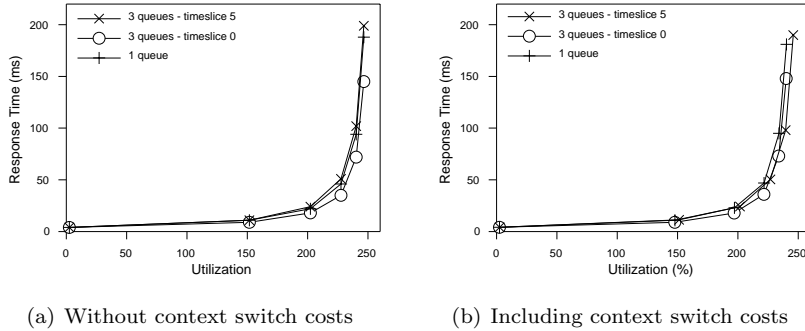


Figure 8: The effects of scheduling strategies

are presented. Finally, the simulations with and without optimizations are presented in five experiments. The simulations compares the performance of the commit protocols presented R2PC, C2PC, R1PC, C1PC and CCP, using each of the optimization, i.e. EA, DBC, DPC, DPC-P and none.

For all simulations there is no thrashing. This is because the amount of work that needs to be done does not increase when the number of transactions get high. However, if the arrival rate is higher than the maximum throughput for each protocol, queues will build up and the response time suffers. In that case the throughput still stays close to the maximum throughput.

5.1 Experiment 1: Scheduling

The scheduling of processes influences the priority of operations. This affects both the response time and the throughput. Generally, increasing the number of context switches improves response time, but hurts throughput.

This experiment is designed to compare three different scheduling strategies. The first is keeping one queue for each cpu. The queue is used by all processes, and requests in the queue is served on a first-come first-served basis. The second and third scheduling strategies keep a total of three queues as shown in Figure 7(b): One queue for incoming messages, one queue for tasks for the TM, and one queue for outgoing messages. For the second strategy, each process executes only one operation between each context switch, while the third gives each process a maximum timeslice of 5 milliseconds before it is switched out. For both, the OS process executes send and receive operations in an interleaving fashion, starting with send.

Two types of simulations were performed. One where the cost of the context switch is set to zero to examine the behavior of the priority in isolation, and one with normal cost for context switching. During the latter, the one queue case performs a context switch between each operation.

Figure 8 shows the results of these simulations. Figure 8(a) shows that more time is spent in the queues using the three queues and 5 ms timeslice, with the single queue slightly better. Performing only one from each queue gives a better response time, as the queue for the TM and sending will almost always be empty, thus prioritizing completing requests as soon as possible. For low utilization, Figure 8(b) indicates the same. The three queues and timeslice 5 achieves better throughput than the others, since it performs less context switches, but has a

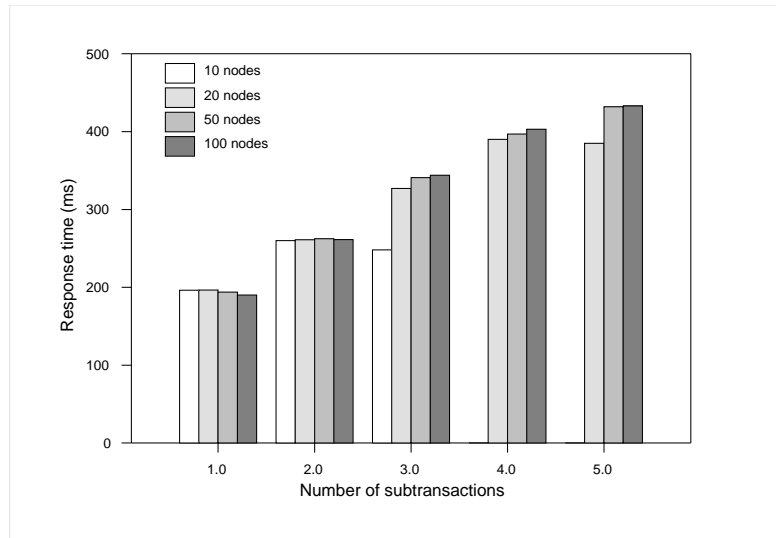


Figure 9: The number of participants versus the number of nodes at 97.5% utilization

worse response time.

5.2 Experiment 2: Number of Participants and Nodes

The number of participants for each transaction affects the throughput of the system. Clearly, adding more participants leads to an increase in the demand on the system, causing a decrease in throughput (here measured as transactions per seconds per node, tps/node).

For this experiment, the third scheduling policy (Timeslice 5) from Experiment 1 is used, and simulations are performed for one, two, three, four and five participants. The utilization of the system for these simulations is 97.5%.

As seen in Figure 9 the response time increases for an increasing number of participants as expected. However, the figure also indicates an increasing response time for an increasing number of nodes. Furthermore, the gap between a few nodes (10 and 20) and many nodes (50 and 100) grow larger when extra nodes are added. This effect is caused by simple statistics. The response time of a transaction is heavily bound by the slowest participant. Thus, if only one of the participating nodes are overloaded, the transaction is delayed. For more simulated nodes, there is a larger probability that there exists an overloaded node, than for fewer nodes. Hence, the response time increases for increasing numbers of nodes.

5.3 Experiment 3: Number of Nodes and Various Protocols

This experiment looks at the impact of varying the number of node and compares R2PC, R1PC, C2PC, C1PC and CPP to a baseline protocol called *Distributed Processing, Centralized Commit*, DPCC [12]. DPCC executes the distributed transactions as usual, but skips the distributed commit processing,

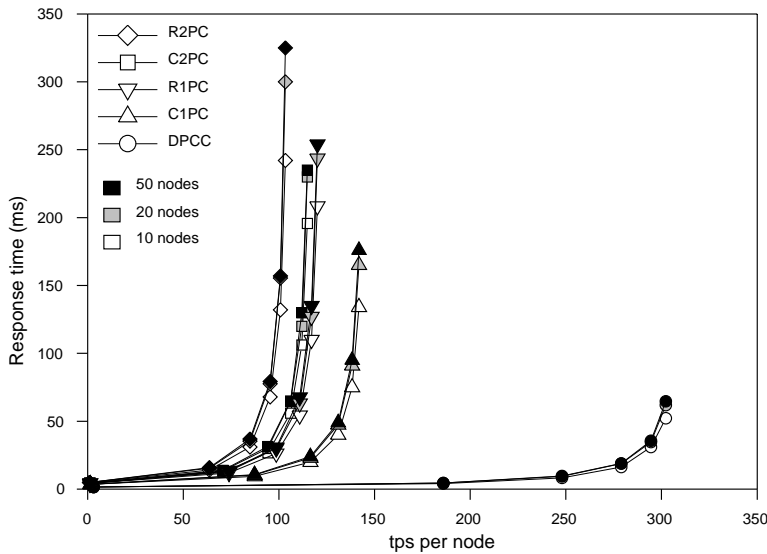


Figure 10: Simulation of 10, 20 and 50 nodes for various protocols

i.e. there no commitment protocol is executed. Thus, it is not a fault-tolerant protocol, but serve as a baseline, against which the other can be evaluated.

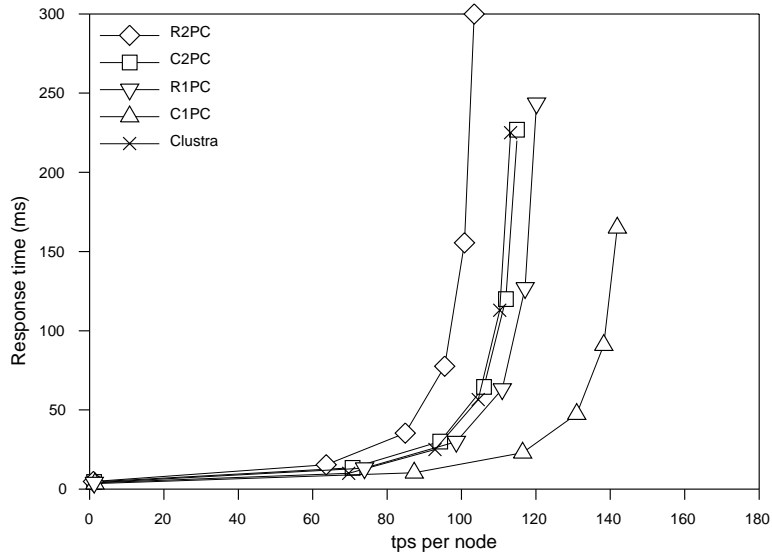
The simulations was performed while varying the number of nodes. Figure 10 shows the results. Black, grey and white markers are 50, 20 and 10 nodes, respectively, while circles, squares and diamonds are DPCC, C2PC and R2PC, respectively. Triangles pointing down are R1PC while the ones pointing up are C1PC. This experiment verifies the result from Experiment 2: An increasing number of nodes increases the response time. Comparing C2PC and R2PC to DPCC, it is shown that distributed commit processing reduces the throughput and increases the response time significantly for short update transactions.

5.4 Experiment 4: No Optimizations

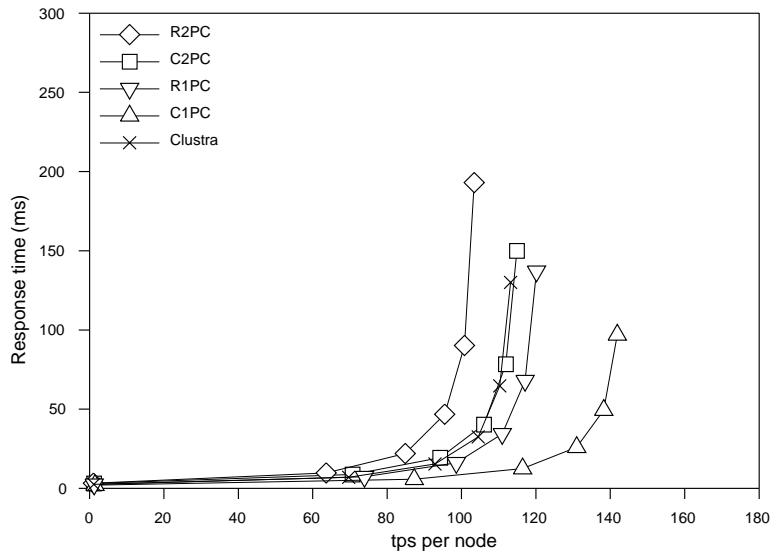
The response time of a transaction is the time from a client starts to send the request to the system, until it receives a reply. For standard transaction processing, the system does not reply to the client until the commit processing has completed. This experiment compares the response time of CCP, C2PC, R2PC, C1PC and R1PC without using the optimizations.

The results from 20 nodes shown in Figure 10 is drawn again in Figure 11(a). The simulations with 10 and 50 nodes are removed to avoid cluttering it.

As can be seen from Figure 11(a), C2PC gives a maximum throughput of around 11% more than R2PC and 2% more than CCP. C1PC results in a 38%, 26%, 25% and 19% increase in throughput compared to R2PC, CCP, C2PC and R1PC, respectively. Also, a significant improvement in response time is achieved.



(a) Completion time without any optimizations



(b) Response time using the Early Answer optimization

Figure 11: Simulating commit protocols over 20 nodes

5.5 Experiment 5: Early Answer

This experiment uses the Early Answer (EA) optimization applied to an experiment otherwise equal to Experiment 4. Figure 11(b) shows the results. They can be compared to the results without any optimizations in Figure 11(a). The maximum throughput is the same as for the simulations in Experiment 4, since the total amount of work is the same. The average response times, however, is significantly reduced by 32-40% for C2PC and R2PC, 40-46% for C1PC and R1PC, and 38-44% for CCP.

5.6 Experiment 6: Dynamic Backup Coordinator

The Dynamic Backup Coordinator approach (DBC) was simulated in conjunction with EA. The results are shown in Figure 12(a). Compared to just using the early answer optimization (Figure 11(b)) this reduces the response time and increase the maximum throughput with about 4% and 7% for C1PC and C2PC, respectively. This reduction is caused by avoiding some processing at and communication between the joint backup participant and backup coordinator node. However, this optimization does not apply to R2PC, R1PC and CCP since there is no direct communication between the backup participants and the backup coordinator.

5.7 Experiment 7: Dynamic Primary Coordinator

The Dynamic Primary Coordinator approach (DPC) was used in conjunction with EA for these simulations. Figure 12(b) shows the results of the simulations. Compared to only using EA, the response time increases by 20 - 30% for utilizations up to 60%, but is more effective for higher utilizations. The reason for the increase at lower utilizations, is the need for the processing at the node forwarding the transaction. The maximum throughput is increased by 20 - 25% by using DPC in addition to EA. CCP has a better response time than C2PC and R1PC for utilizations below 80%, but because of higher total overhead, the maximum throughput is lower. Again, R2PC is the worst performer and C1PC is the best.

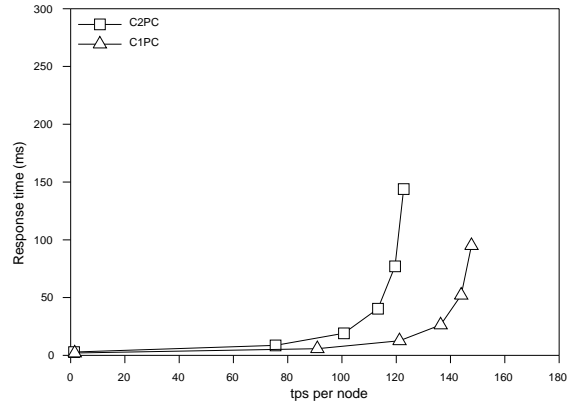
5.8 Experiment 8: Piggybacking YouAreCoord

The Dynamic Primary Coordinator and Piggybacking YouAreCoord approach (DPC-P) was used in conjunction with EA for these simulations. Figure 12(c) shows the results. The results from the DPC-P simulation is marginally better than the results from DPC. The maximum throughput increases about 1% for each of the protocols and the response time is slightly reduced since there is less delay before sending out the work requests to the participants. The general form of all plots is nearly identical to that of DPC in Figure 12(b).

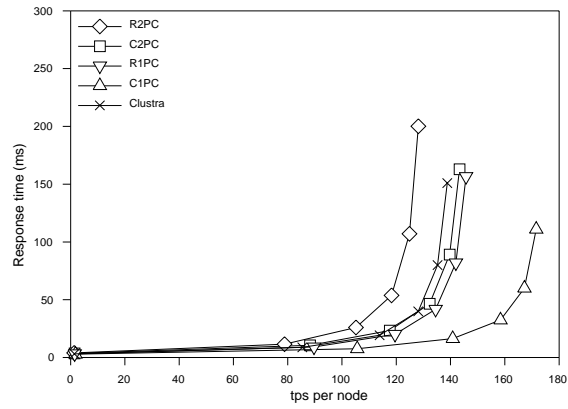
5.9 Comparing Optimizations

Figure 13 shows the improvement for each type of protocol by using the enhancers described in Section 3.5. All of them display the same pattern. Not surprisingly, the simulations without optimizations have the least throughput, followed by EA, DBC, DPC and finally DPC-P at the highest end. For R2PC in Figure 13(a), R1PC in Figure 13(c) and CCP in Figure 13(e), there is no plot for DBC simply because there is no communication saved since there is no direct communication between the backups. For DPCC in Figure 13(f), both EA and DBC is not plotted as it would be the same as the transaction completion plot.

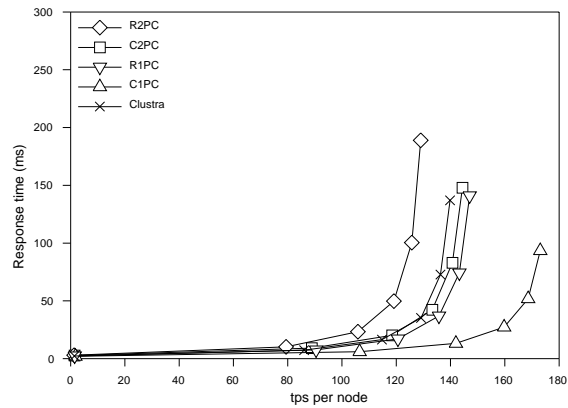
The response times are generally improved by one-third in EA compared to the one without optimizations. Where DBC shows improvement over EA (C2PC and C1PC in Figures 13(b) and 13(d)), it is only marginally faster. Compared to EA, DPC increases the response time for throughputs up to 80 tps/node by 20 - 35% because of the extra costs of forwarding the transactions. For higher



(a) The DBC optimization



(b) The DPC optimization



(c) The DPC-P optimization

Figure 12: The average response times using various enhancers

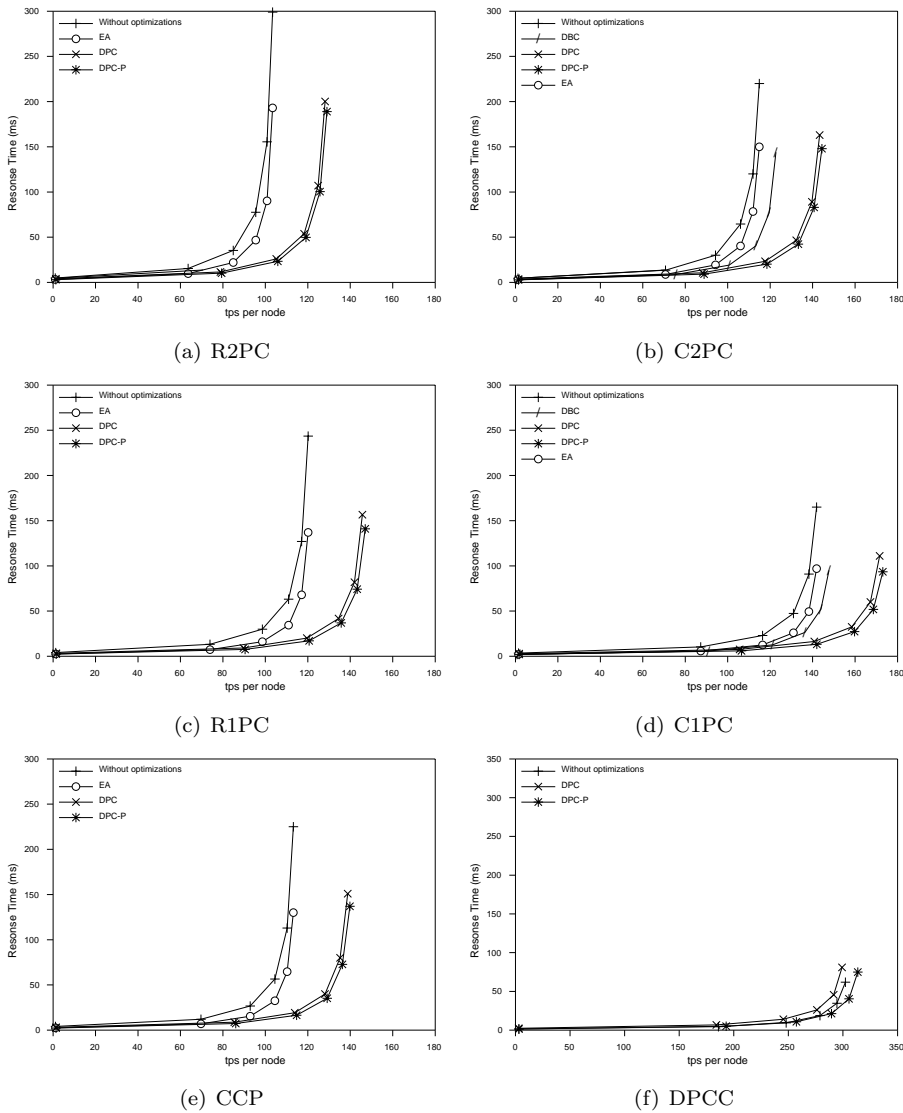


Figure 13: The average response times using the performance enhancers on each of the protocols

throughputs, DPC is faster since it tolerates more transactions. Contrary to DPC, DPC-P improves the response times for low throughputs as well. The difference is clearly visible in all graphs from around 60% throughput for EA.

For a utilization of 97.5%, shown as the highest point in all graphs in Figure 13, it is clear that DPC-P provides the best response time for all protocols.

As shown in Figure 13(f), DPCC is the only one with loss in throughput when using DPC compared to no optimization, EA and DBC. This is because the extra processing at the start by forwarding the transactions is never made up for by the commit protocol. However, using DPC-P, a slight performance increase is achieved.

The differences in throughput between executing transactions without using

optimizations and transactions using DPC-P are substantial. Looking at the differences in maximum throughput for the optimizations shows the improvement. R2PC and C2PC improves by 25%. C1PC improves by 22%, while R1PC and CCP improves by 23%. Comparing the best, C1PC - DPC-P, to the worst, R2PC and the existing MM protocol, CCP - EA, an increase of throughput of 68% and 45% is achieved.

5.10 Response Time Demands

For real-time databases, a frequent demand is to have 95% of the transactions to respond within a few milliseconds. Figure 14 shows the response time for the 95% quickest of the transactions for all protocols using the best optimization, DPC-P. This means that only 5% of the transaction responded slower for each of the plots. In Figure 14(a) and 14(b), the 10 ms and 5 ms response time, respectively, is marked with a dotted line, and arrows mark the limit in throughput for each of the protocols.

The arrows in Figure 14(a) shows that C1PC can tolerate approximately 95%, 55%, 40% and 35% more than R2PC, C2PC, CCP and R1PC, respectively, before crossing the 10 ms limit for more than 5% of the transactions. Figure 14(b) illustrates the improvements for C1PC versus R2PC, C2PC, CCP and R1PC for a 5 ms response time limit, are about 190%, 150%, 65% and 60%, respectively. This indicates that the relative performance of the C1PC protocol over the others are increasingly getting better as a system response time requirements get more stringent.

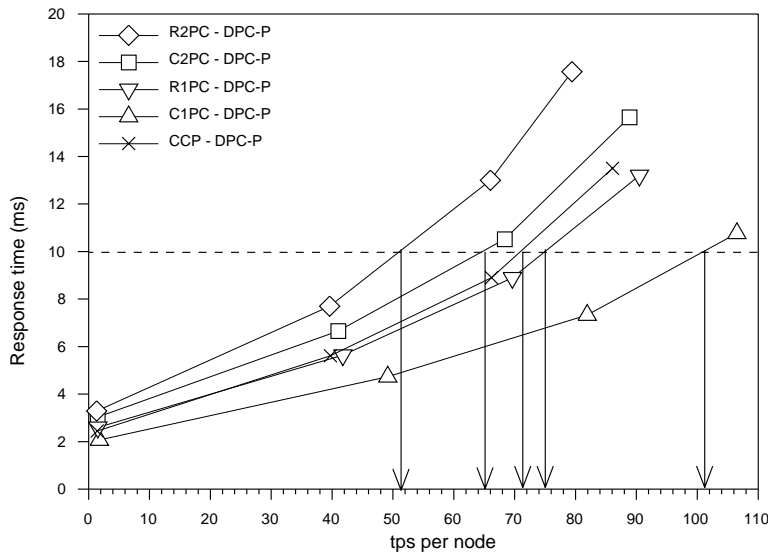
5.11 Distribution

The response time distribution for C1PC using DPC-P at 60% utilization (106 tps/node) is shown in Figure 5.11. Each response time have been rounded to the nearest integer value and the number of transactions for each are counted. Almost all replies before 15 ms, and around 94% replies within 10 ms. This fits well with the plot for C1PC in Figure 14, show that for a load of 101 tps/node, 95% of the transactions to reply within 10ms.

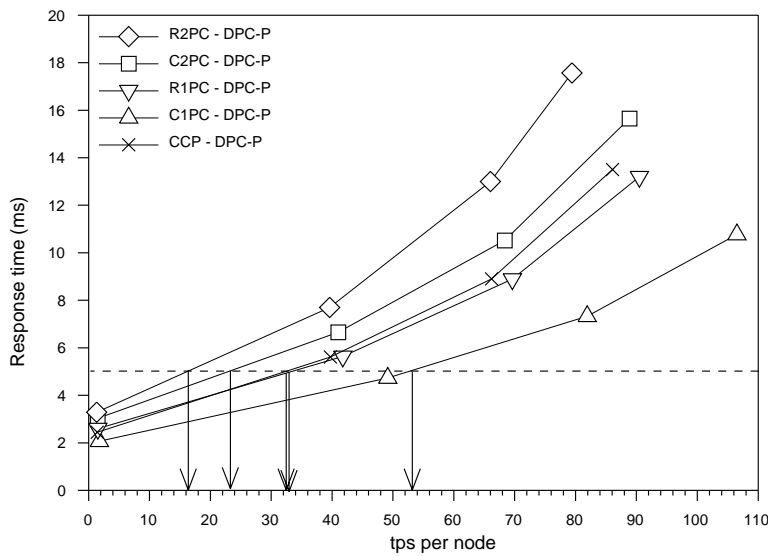
6 Analysis

To verify the results from the simulations, an analytical model of the system was constructed. The transaction workload is split into individual operations (or classes), such as `BeginTxn`, `DoWork`, `Commit`, `Send`, etc. Also, there is an unlimited number clients of the system. This gives a multiple class, open queuing network.

A Poisson process generates the requests with exponentially distributed inter-arrival times. The number of operations per transaction are counted for each type of protocol to give the relative frequency of each of the classes. The load added by context switches are treated as a separate workload class. Its frequency is found by using the output from the simulations and counting the number of context switches performed for each combination of protocol and utilization.



(a) 10 milliseconds limit



(b) 5 milliseconds limit

Figure 14: The maximum response time for the 95% quickest transactions for the DPC-P optimization. 20 nodes are simulated and the graphs are cut off at 60% utilization

Assuming uniformity across the nodes, single-server queueing theory [31] can be used. This is a single resource, with a single queue in front. The Poisson process generates at an arrival rate given by the sum of the arrival rates from each class for the given protocol. The steady-state solutions are derived using results from $M/G/1$ queues [31]. The resulting total residence time is then weighed against the number of serial requests of each class. To find the slowest response time for the three subtransactions, numerical Laplace inversion of waiting times

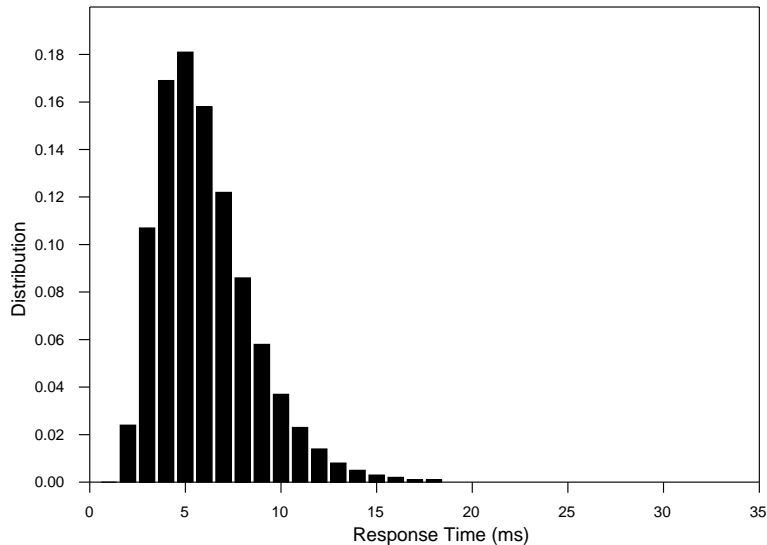


Figure 15: Distribution of response times for C1PC with the DPC-P optimization at 60% utilization

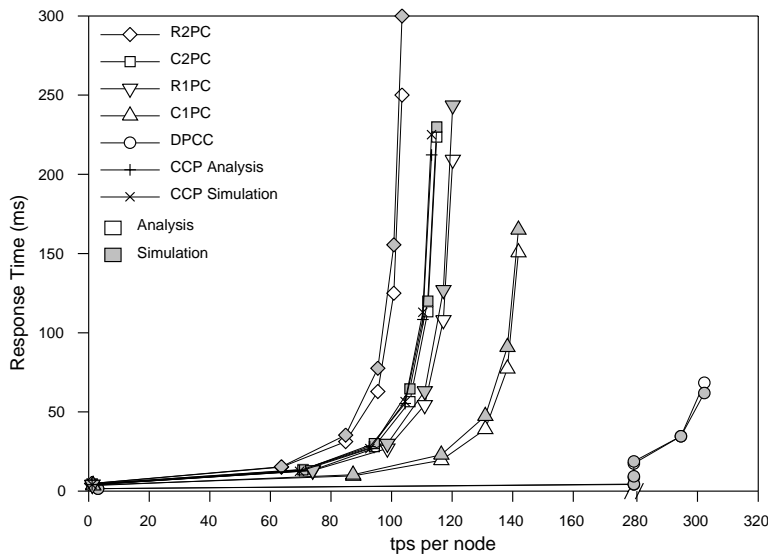
of $M/D_N/1$ queues [32] is used.

Figure 16 shows a comparison of the analyses and simulations of 20 nodes. The first is without any optimization and the second is the best performing, DPC-P. There are some discrepancies between the analysis and simulation. These are caused by some factors from the simulations that are not included in the analysis: (1) The priority as introduced by the scheduling policy (shown by Experiment 1) and (2) the number of subtransactions and nodes (shown by Experiment 2 and 3). The first slightly increases the simulation response time since the analysis uses a scheme close to the one queue approach in Section 5.1. For the second, when varying the number of nodes, the differences in response time are quite large as seen in Figure 10. The response time analysis for all protocols are between the simulation of 10 and 50 nodes. Despite the differences shown in this analysis, the shape of the analysis plots seem to verify the general shape of the simulation plots for high utilization.

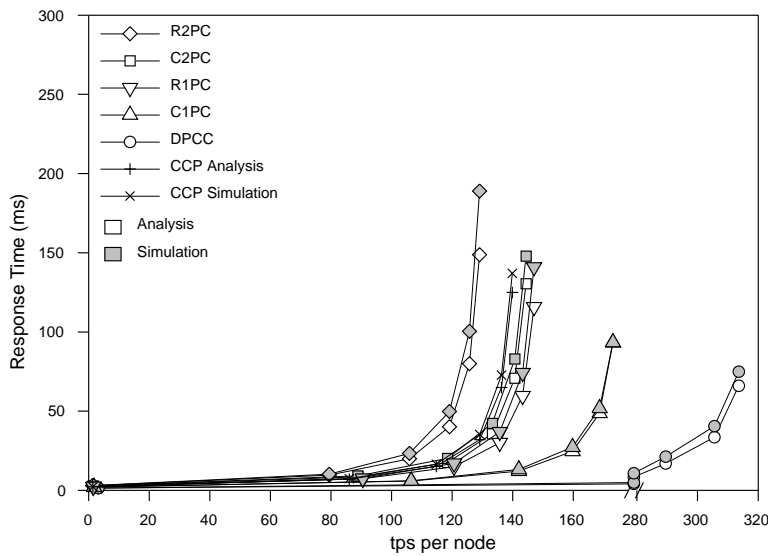
Figure 17 compares the simulation and analysis for utilizations below 60%, except for DPCC which is plotted until 30%. This is in the range where a system normally operates and the maximum difference here are less than 7%. Thus, the simulations are clearly well suited to the analysis for normal operating conditions.

7 Evaluation and Conclusion

This article has presented a performance evaluation of various main-memory commitment protocols by simulation. The simulation results was verified analytically. When applicable, the one-phased protocols perform better than the two-phase protocols, with C1PC as the best one. CCP is better than R1PC at throughputs below 60 tps/node. If a two-phase protocol is required, the results show that a significant performance gain can be achieved by using C2PC rather



(a) Without optimizations

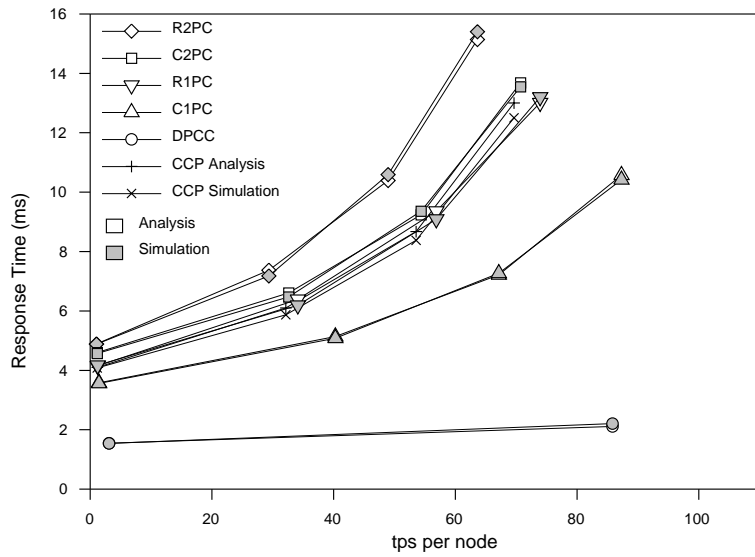


(b) With DPC-P

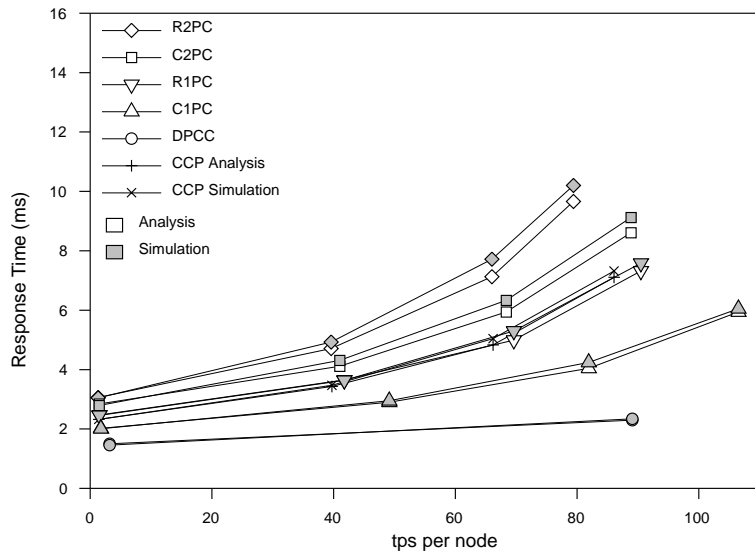
Figure 16: Comparing analysis and simulations for protocols at high utilization than R2PC.

The dynamic optimizations improves the response time and throughput even further. DPC-P improves the throughput by 22-25% for all protocols compared to not using any optimizations. The best performing combination of protocol and optimization, was found to be C1PC using the DPC-P optimization. It supports nearly 70% more transactions per seconds than the worst, R2PC without optimizations, and even reduce the minimum possible response time with almost 60%.

Using the response time requirement stating that 95% of the transactions



(a) Without optimizations



(b) With DPC-P

Figure 17: Comparing analysis and simulations for protocols at low utilization responds within a time limit of 10 ms, C1PC was found to support 95%, 55%, 40% and 35% more tps than R2PC, C2PC, CCP and R1PC, respectively, all using the DPC-P optimization.

For further work a full scale implementation of the best protocol and optimizations should be incorporated in an existing main-memory system. It would show the effect of these protocols for real world applications. Also, an analytical model where the effects of varying the number of nodes is clear should be developed.

References

- [1] Garcia-Molina, H., Salem, K.: Main memory database systems: An overview. *IEEE Transactions on Knowledge and Data Engineering* **04** (1992) 509–516
- [2] DeWitt, D.J., Katz, R.H., Olken, F., Shapiro, L.D., Stonebraker, M.R., Wood, D.: Implementation techniques for main memory database systems. In: *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, New York, NY, USA, ACM Press (1984) 1–8
- [3] Cristian, F.: Understanding fault-tolerant distributed systems. *Communications of the ACM* **34** (1991) 56–78
- [4] Gray, J.: The transaction concept: Virtues and limitations (invited paper). In: *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, IEEE Computer Society (1981) 144–154
- [5] Heine Kolltveit and Svein-Olaf Hvasshovd: The Circular Two-Phase Commit Protocol. In: *Database Systems for Advanced Applications, 12th International Conference, DASFAA. Lecture Notes in Computer Science*, Springer (2007)
- [6] Inseon Lee and Heon Y. Yeom and Taesoon Park: A New Approach for Distributed Main Memory Database Systems. *IEICE Transactions on Information and Systems* **E87D** (2004) 196–204
- [7] Boutros, B.S., Desai, B.C.: A two-phase commit protocol and its performance. In: *DEXA '96: Proceedings of the 7th International Workshop on Database and Expert Systems Applications*, Washington, DC, USA, IEEE Computer Society (1996) 100
- [8] Y.J. Al-Houmaily and R. Conticello and J.Pike and P.K. Chrysanthis: Performance of Five Atomic Commit Protocols in Gigabit-Networked Database Systems. In: *Proc. of 11th Int'l Conference on Parallel and Distributed Computing Systems*. (1998) 29–36
- [9] Al-Houmaily, Y.J.: Commit processing in distributed database systems and in heterogeneous multidatabase systems. PhD thesis, Pittsburgh, PA, USA (1997)
- [10] Gupta, R., Haritsa, J., Ramamritham, K.: Revisiting commit processing in distributed database systems. In: *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, New York, NY, USA, ACM Press (1997) 486–497
- [11] P. K. Chrysanthis and G. Samaras and Y. J. Al-Houmaily: 13. In: *Recovery and Performance of Atomic Commit Processing in Distributed Database Systems*. Prentice Hall (1998) 370–417
- [12] Haritsa, J.R., Ramamritham, K., Gupta, R.: The {PROMPT} real-time commit protocol. *IEEE Trans. Parallel Distrib. Syst.* **11** (2000) 160–181

- [13] Samaras, G., Kyrou, G.K., Chrysanthis, P.K.: Two-phase commit processing with restructured commit tree. In: Proc. of PCI. (2003) 82–99
- [14] Lee, I., Yeom, H.Y.: A single phase distributed commit protocol for main memory database systems. In: IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium, Washington, DC, USA, IEEE Computer Society (2002) 44
- [15] Lee, I., Yeom, H.Y.: A fast commit protocol for distributed main memory database systems. In: ICOIN '02: Revised Papers from the International Conference on Information Networking, Wireless Communications Technologies and Network Applications-Part II, London, UK, Springer-Verlag (2002) 691–702
- [16] Hvasshovd, S.O., Torbjørnsen, Ø., Bratsberg, S.E., Holager, P.: The Clus-tRa telecom database: High availability, high throughput, and real-time response. In: Proc. of VLDB. (1995)
- [17] Gray, J.: Notes on data base operating systems. In Flynn, M.J., Gray, J., Jones, A.K., Lagally, K., Opderbeck, H., Popek, G.J., Randell, B., Saltzer, J.H., Wiehle, H.R., eds.: Operating Systems, An Advanced Course. Volume 60 of Lecture Notes in Computer Science., Springer (1978) 393–481
- [18] Liu, M.L., Agrawal, D., Abadi, A.E.: The performance of two phase commit protocols in the presence of site failures. *Distrib. Parallel Databases* **6** (1998) 157–182
- [19] Mohan, C., Lindsay, B., Obermarck, R.: Transaction management in the R* distributed database management system. *ACM Trans. Database Syst.* **11** (1986) 378–396
- [20] Stamos, J.W., Cristian, F.: Coordinator log transaction execution protocol. *Distributed and Parallel Databases* **1** (1993) 383–408
- [21] Al-Houmaily, Y., Chrysanthis, P.: Two-phase commit in gigabit-networked distributed databases (1995)
- [22] Agrawal, R., Carey, M.J., Livny, M.: Concurrency control performance modeling: Alternatives and implications. *ACM Trans. Database Syst.* **12** (1987) 609–654
- [23] Thomasian, A.: Concurrency control: methods, performance, and analysis. *ACM Comput. Surv.* **30** (1998) 70–119
- [24] Nicola, M., Jarke, M.: Performance modeling of distributed and replicated databases. *IEEE Transactions on Knowledge and Data Engineering* **12** (2000) 645–672
- [25] Mohan, C., Lindsay, B.G.: Efficient commit protocols for the tree of processes model of distributed transactions. In: PODC. (1983) 76–88
- [26] Lamson, B.W.: Atomic transactions. In: Distributed Systems - Architecture and Implementation, An Advanced Course, London, UK, Springer-Verlag (1981) 246–265

- [27] Abdallah, M., Guerraoui, R., Pucheral, P.: One-phase commit: Does it make sense? In: Proc. of ICPADS, Washington, DC, USA (1998)
- [28] Stonebraker, M.: Concurrency control and consistency of multiple copies of data in distributed ingres. *IEEE Trans. Software Eng.* **5** (1979) 188–194
- [29] Bernd Page and Wolfgang Kreutzer: *The Java Simulation Handbook. Simulating Discrete Event Systems with UML and Java.* Shaker Verlag (2005)
- [30] Loland, J., Hvasshovd, S.O.: Online, non-blocking relational schema changes. In: *Advances in Database Technology – EDBT 2006.* Volume 3896 of *Lecture Notes in Computer Science.*, Springer-Verlag (2006) 405–422
- [31] Jain, R.: *The Art of Computer Systems Performance Analysis.* Wiley & sons (1991)
- [32] John F. Shortle and Martin J. Fisher and Percy H. Brill: Waiting Time Distribution of $M/D_N/1$ Queues Through Numerical Laplace Inversion. *INFORMS Journal on Computing* (to appear)