Efficient High Availability Commit Processing

Heine Kolltveit and Svein-Olaf Hvasshovd

Abstract—Distributed transaction systems require an atomic commitment protocol to preserve ACID properties. A commit protocol should add as little overhead as possible to avoid hampering performance. In this paper, dynamic coordinators are introduced. In main memory primary-backup systems, the approach significantly reduces the time spent during commit processing. The performance of such protocols must be properly evaluated to give system developers the information needed to make an educated choice between them. Thus, simulation results, verified by statistical analysis, are presented. The simulation results show that the performance can be significantly boosted by using optimizations and protocols especially designed for high availability main memory systems.

I. INTRODUCTION

Computer systems are getting increasingly more complex, which put increasing pressure on the performance of each of the components. For databases, being a central building block, it is crucial that they are fast and reliable. Declining main memory (MM) prices and increasing storage capacity have contributed to the advent of MM databases and applications. Contrary to traditional disk resident databases, main memory databases (MMDBs) avoid slow disk accesses by permanently storing all data in main memory [1]. This speeds up data access, since access to main memory is much faster than access to 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 is 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 data nor log resides on disk. The log and data can be made persistent by replicating it between processes at separate nodes. This paper is in the context of persistent pure main memory databases.

Messages are sent to replicate data between processes. 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.

Tailored protocols are needed to enable MM systems to exploit the potential performance advantages of their nature. A transactional system, such as a database, requires 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], [6], none are targeted at pure MM protocols.

This paper is motivated by the observation that 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, it results in unnecessary overhead. Fewer nodes involved in commit processing means less overhead. In this paper, dynamic coordinators are used to enhance the performance of main memory commit protocols. In addition, there is a 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 main contributions of this paper are novel optimizations for MM commit protocols and analytically verified simulation results comparing the relative performance of MM commit protocols with and without these optimizations. Also, a framework for analysing the performance impact for various MM commit protocols and enhancements is outlined. The results of such an evaluation can favorably be used by system developers to improve the performance of MM applications, e.g., in a shared-nothing fault-tolerant DBMS like ClustRa [15].

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

The rest of the paper is organized as follows. Section II gives an overview of related work. Section III presents the commit protocols discussed in this paper, while Section IV introduce the new optimizations using dynamic coordinators. Section V outlines the simulation model and parameters used for the simulations in Section VI. The simulations are compared to a statistical analysis in Section VII, while Section VIII gives some concluding remarks.

II. RELATED WORK

A. Optimizations

Several 2PC-based modifications where performance issues are handled exist [16]. Presumed commit and presumed abort [17] both avoid one flushed disk write, by assuming that a non-existent log record means that the transaction has committed or aborted, respectively. Transfer-of-commit, lazy commit and read-only commit [18], sharing the log [17], [19] and group commit [20], [21] are other optimizations. An optimization of the presumed commit protocol [22] reduces the number of messages, but requires the same number of forced disk writes.

Optimistic commit protocols are designed to give better response time during normal processing, but will need extra recovery after failures or aborts. They release locks when the
transaction is prepared, but must be able to handle cascading aborts by using semantic knowledge [23]. PROMPT [12] uses optimistic locking in the sense that locks can be lent to other transactions after the participant has voted yes. A transaction that lends locks will not reply to the request until the locks are fully released by the previous transaction, and only one transaction at a time can lend a lock. This approach avoids cascading aborts while it may yield better performance because of increased concurrency.

One-phased commit protocols have also been proposed [19], [24], [25], [14], [26], [15]. These are based on the early prepare or unsolicited vote method by Stonebraker [27] where the prepare message is piggybacked on the last operation sent to a participant. In this way, the voting phase is eliminated. However, these approaches may inflict strong assumptions and restrictions on the transactional system [25].

One existing approach combines optimistic commit and replication [28]. A replicated group of commit servers is used to keep the log records not yet written to the log by the participant available, thus ensuring resilience to failures. This approach uses multicast and has the same latency as 2PC, but requires more messages to be sent.

Commit protocols for replicated main-memory systems have been introduced by Kolltveit and Hvashovd [5] and in the context of ClustRa [15]. These are presented in greater detail in Section III.

B. Performance evaluations

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 [29].

Even if 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 rerequests before giving up are presented by Boutros and Desai [7]. Also, Liu, Agrawal and Abbadi [30] 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 for a completely centralized system, and the other for a distributed transaction processing and centralized commit processing, have been performed [10]. The study considers 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 [17]) and two one-phased protocols (Coordinator Log [26] and Implicit Yes-Vote (IYV) [31]). 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 [32], [6]. However, this approach assumes synchronous logging to disk, severely degrading the commit processing 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 [33], [34], [35]. They are mostly focused on other aspects of transaction processing, mainly concurrency control, and the statistical analysis of different types of commit processing have been overlooked.

III. COMMIT PROTOCOLS

This section presents commit protocols for main-memory primary-backup systems. They are introduced by outlining the standard commit protocols that have influenced them.

A. Two-Phase Commit

The Two-Phase Commit Protocol (2PC) [29], [36] consists of two phases: The voting phase and the decision phase. During the voting phase the votes are collected from the transaction participants by the transaction coordinator: It sends out a Prepare message. Then, before replying with a Vote message, each participant makes its vote durable by forcing to stable storage [37]. 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, if the coordinator agrees with a positive outcome, 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.

B. Replicated Two-Phase Commit

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. Fig. 2(a) illustrates this. The small arrows between each primary-backup pair is the logging. The rest of the protocol remains unchanged.

Fig. 1. Legend for Fig. 2 and Fig. 3

client process
primary replica
primary-backup pair
backup replica
messages
coordinator C
participant P

Fig. 1. Legend for Fig. 2 and Fig. 3
C. Circular Two-Phase Commit

The Circular Two-Phase Commit Protocol (C2PC), as presented by the authors [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 Fig. 2(b) 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.

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 have been sent, 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].

D. 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 [25]. The oldest is the Unsolicited Vote protocol (UV) outlined by Stonebraker [27] where Prepare is piggybacked on DoWork. Thus, the voting phase is included in the transaction processing. 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 [5].

1) Circular One-Phase Commit: The Circular One-Phase Commit Protocol (C1PC) is a one-phased version of C2PC, based on UV, as presented by the authors [5]. Fig. 2(c) illustrates the protocol. 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.

2) ClustRa Commit Protocol: The commit protocol used in ClustRa [15], uses a one-phased main-memory commit protocol. Fig. 2(d) 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 receive a Commit, and reply with Committed. The successful completion of the commit processing is then persistently stored at the backup coordinator, before the transaction is completed and forgotten.

The Early Answer optimization, EA, used by ClustRa is applicable to the other protocols as well. Generally, 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. For 2PC, the reply can be given once the decision is saved to disk and for C2PC, R2PC, C1PC and R1PC the reply can be sent as soon as both the primary and backup coordinators know the outcome of the transaction. Hence, the time to execute the decision phase of the commit processing is not included in the response time. Thus, the response time of a transaction, as seen by clients, is reduced at no extra costs.

In this paper, we refer to the ClustRa commit protocol without the early answer optimization as CCP to keep with the syntax of the other optimizations. The protocol as presented in [15] includes, however, the early answer optimization and is labeled CCP-EA from here on.

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

IV. Dynamic Coordinators

When executing distributed transactions, the number of participating nodes should be kept at a minimum. This can be achieved by dynamically assigning the coordinator tasks to one of the participants. This section present some new 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.

A. Dynamic Backup Coordinator

One way to reduce the overhead is to make one of the backup participants the backup coordinator of the transaction. This optimization is called the Dynamic Backup Coordinator, DBC. This will reduce the overhead for C1PC and C2PC, but not for the general commit protocol. DBC is illustrated in Fig. 3(b). 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 separate ones. This saves two messages, one prepare task and one commit task. Clearly, since it is a fixed reduction of cost, this approach yields a better relative improvement for few participants, than for many. In Fig. 3(b) and during the simulations, DBC are executed with EA.

B. Dynamic Primary Coordinator

Another optimization is to forward the coordination of each transaction to a primary participant. Thus, one of the primary participants 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 Fig. 3(c). The performance gain is a complete round of commit processing for one node, minus the extra overhead of forwarding the transaction. As with DBC, the relative improvement is increased as the number of participants decreases.

DPC is applicable to all commit protocols described in Section III. In Fig. 3(c) and during the simulations DPC use EA.

C. 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 Fig. 3(d). 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. As with DBC and DPC, the relative improvement is greater for few participants. In the figure and during the simulations DPC-P are executed with EA.

V. 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 had to be developed. It has been implemented using Desmo-J, a framework for discrete-event modelling and simulation [38].

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 Java main memory database prototype developed by Løland and Hvasshovd [39].

The distributed transaction processing (TP) system considered in this paper is composed of \( N \) nodes connected through a communication network, as shown in Fig. 4(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 is associated with a coordinator, which is responsible for execution and termination. 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 participant TMs.

The distribution of processing and data is transparent to the transaction clients. 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 [17]. However, this paper discusses only trees that are one level deep. Transactions are assumed to be precompiled, and all nodes know 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 shown in Fig. 4(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.

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 $\mu$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.

An open simulation model is used. The utilization of the system were varied from 1% - 97.5% to see the impact on the system performance. To be able to set the utilization for the system were varied from 1% - 97.5% to see the impact on the system performance. To be able to set the utilization for the system

Each transaction consists of three subtransactions, each of which containing a simple update operation. Thus, during transaction processing each subtransaction makes one visit to its respective participant, and returns. The subtransactions are performed in parallel. The coordinator and participants are chosen randomly and uniquely for each transaction. Since none of them are the same, 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 the load of a distributed main memory database. Also, small messages have a very short transit 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, as to send a message.

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) take 700 $\mu$s, medium operations (BeginTxn, Prepare, Abort and Commit) take 150 $\mu$s and short operations (WorkDone, Vote, Aborted, Committed) takes 50 $\mu$s.

As the primary focus of this paper is the performance of commit protocols, the database internals are not modelled. A very large number of records is assumed, giving negligible data contention, hence, lock waiting effects have been ignored.

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 with random seed generators were made.

Table I summarizes the simulation parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Model</td>
<td>Open</td>
</tr>
<tr>
<td>Context Switch</td>
<td>3.5 $\mu$s</td>
</tr>
<tr>
<td>Timeslice</td>
<td>5 ms</td>
</tr>
<tr>
<td>Send Message</td>
<td>40 $\mu$s</td>
</tr>
<tr>
<td>Receive Message</td>
<td>80 $\mu$s</td>
</tr>
<tr>
<td>Long Operations</td>
<td>700 $\mu$s</td>
</tr>
<tr>
<td>Medium Operations</td>
<td>150 $\mu$s</td>
</tr>
<tr>
<td>Short Operations</td>
<td>50 $\mu$s</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>200 s</td>
</tr>
<tr>
<td>Capture Time</td>
<td>160 s (40 - 200)</td>
</tr>
<tr>
<td>Simulation runs</td>
<td>10</td>
</tr>
<tr>
<td>Subtransactions</td>
<td>3</td>
</tr>
<tr>
<td>Transactions</td>
<td>Single tuple update</td>
</tr>
<tr>
<td># of simulated nodes</td>
<td>20</td>
</tr>
<tr>
<td>CPU Utilization</td>
<td>Variable</td>
</tr>
</tbody>
</table>
optimization. Third, the distribution of the response times is investigated.

A. Average Response Time

Fig. 5 plots the response time versus the throughput for each combination of protocol and optimization. No optimizations is plotted as diamonds, EA as triangles pointing down, DBC as triangles pointing up, DPC as squares and DPC-P as circles. Fig. 5(a) shows C2PC in black and R2PC in white, and Fig. 5(b) shows C1PC in black, R1PC in white and CCP in grey.

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. Comparing the protocols using no optimizations, 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.

The Early Answer (EA) optimization replies earlier in the transaction processing. The maximum throughput is the same as not using any optimization since the total amount of work is the same. The average response times, however, are significantly reduced by 32-40% for C2PC and R2PC, 40-46% for C1PC and R1PC, and 38-44% for CCP. The Dynamic Backup Coordinator approach (DBC) was simulated in conjunction with EA. Compared to just using EA it reduces the response time and increases 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.

The Dynamic Primary Coordinator approach (DPC) was used in conjunction with EA for these simulations. Compared to only using EA, the response times increase 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.

The Dynamic Primary Coordinator and Piggybacking YouAreCoord approach (DPC-P) was simulated with EA. 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 differences in throughput between executing transactions with no optimization or EA 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 in Fig. 5, C1PC - DPC-P, to the worst, R2PC, and the existing MM protocol, CCP - EA, an increase of throughput of 68% and 45% is achieved.

B. Response Time Demands

For real-time databases, a typical requirement is to have 95% of the transactions respond within a given constraint [15]. Fig. 6 shows the maximum response time for the 95% quickest transactions for all protocols using the best performing protocol, C1PC. This means that only 5% of the transaction responded slower for each of the plots. In the figure, the 10 ms and 5 ms response time is marked with a dotted and dashed line, respectively, and arrows mark the limit in throughput for each of the protocols.

The long arrows in Fig. 6 shows that for C1PC - DPC-P tolerate approximately 80% more transactions than not using any optimizations before crossing the 10 ms limit for more than 5% of the transactions. Compared to EA, DBC and DPC the improvement is around 15%. The short arrows illustrates the improvements for DPC-P versus no optimizations, DPC,
EA and DBC for a 5 ms response time limit, are about 250%, 50%, 25% and 25%, respectively. This indicates that the relative performance of C1PC over the others are increasingly getting better as system response time requirements get more stringent. For short response time requirements DPC is performing worse than DBC and EA because of the extra overhead of forwarding the transaction to the correct node.

### C. Distribution

The response time distribution for C1PC using DPC-P at 60% utilization, or 106 tps/node, is shown in Fig. 7. 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 approximately 94% replies within 10 ms. This fits well with the plot for C1PC in Fig. 6, which shows that for a load of 101 tps/node, 95% of the transactions replies within 10 ms.

### VII. Analysis

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

A Poisson process generates 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.

Assuming uniformity across the nodes, single-server queueing theory [40] 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 [40]. 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 of $M/D_N/1$ queues [41] is used.

Fig. 8 shows a comparison of the analyses and simulations of 20 nodes using the best performing enhancer, DPC-P. R2PC is plotted as diamonds, C2PC as squares, R1PC as triangles pointing down, C1PC as triangles pointing up and CCP as circles. The analysis marked as white, while the simulations are shown in black. There are some discrepancies between the analysis and simulation. These are caused by some factors from the simulations that are not covered by this analysis: (1) The priority effect as introduced by the scheduling policy and (2) the number of nodes. The first slightly increases the simulation response time since some of the operations are delayed waiting for the 5 ms timeslice to run out before the
context switch. For the second, when varying the number of nodes, the differences in response time are quite large. The response time analysis for all protocols are between the simulation of 10 and 50 nodes. These issues have been further explored in [42]. 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, as shown in Fig. 8(a).

Fig. 8(b) compares the simulation and analysis for utilizations below 60%. 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.

VIII. EVALUATION AND CONCLUSION

This paper has presented three new optimization techniques in the context of main memory commit protocols. The optimizations have been paired with existing main memory protocols. Also, the protocols and optimizations have been simulated and the results have been verified analytically. The results show that when applicable, one-phased protocols perform better than two-phase protocols, with C1PC as the best one. C2PC was found to be the best two-phased protocol.

The dynamic coordinator optimizations improves the response time and throughput even further. The best performing combination of protocol and optimization, was found to be C1PC 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 and optimizations for real world applications. Also, an analytical model where the effects of varying the number of nodes are shown should be developed. Further, the effects of piggybacking messages on another with the same destination can be investigated.

REFERENCES