A Failure Tolerating Atomic Commit Protocol for Mobile Environments

Stefan Böttcher[1] Le Gruenwald[2] * Sebastian Obermeier[1]

[1]University of Paderborn, Computer Science Fürstenallee 11; 33102 Paderborn; Germany
{stb, so}@uni-paderborn.de

[2]The University of Oklahoma, 200 Telgar street, Norman, OK 73019-3032; USA
ggruenwald@ou.edu

Abstract

In traditional fixed-wired networks, standard protocols like 2-Phase-Commit are used to guarantee atomicity for distributed transactions. However, within mobile networks, a higher probability of failures including node failures, message loss, and even network partitioning makes the use of these standard protocols difficult or even impossible. To use traditional database applications within a mobile scenario, we need an atomic commit protocol that reduces the chance of infinite blocking. In this paper, we present an atomic commit protocol called multi coordinator protocol *(MCP) that uses a combination of the traditional 2-Phase-Commit, 3-Phase-Commit, and consensus protocols for mobile environments. Simulation experiments comparing MCP with 2PC show how MCP enhances stability for the coordination process by involving multiple coordinators, and that the additional time needed for the coordination among multiple coordinators is still reasonable.*

1 Introduction

Applying database technology to a network of mobile devices involves many new challenges like lost connections, node failures, and network partitioning. Especially if there is the need to guarantee atomic transaction commitment, e.g. in the context of distributed databases, peer-to-peer systems, and service oriented architectures, the mobile character of the network makes it difficult to agree on a commit decision and provide it to all participating databases.

Although timeout-based solutions have been proposed (e.g. [7]), timeouts are hard to estimate, especially in a mobile context since transaction execution times and message delivery times vary. This results in an unnecessarily high number of aborts if timeouts are too sensitive and in a long transaction coordination time if timeouts are too long.

Other proposals rely on compensating transactions, which are used to compensate the effect of already committed transactions. However, in mobile networks, where network partitioning makes nodes unreachable but still operational, we cannot assume that the compensating transactions will always reach the desired nodes. Therefore, and because committed transactions can trigger other operations on physically different moving nodes that may disconnect during protocol execution, it cannot be guaranteed that compensation for committed transactions is always possible. Additionally, not all applications have compensatable transactions. Therefore, we consider only the case of non-compensatable transactions in this paper.

To apply distributed database technology in mobile networks, we need a non-blocking atomic commit protocol that not only stabilizes the coordination process, but also reduces the blocking of participating databases, especially if the databases are suspected to frequently disconnect from the network. A complete change of existing database applications and the underlying transactional concepts is a hard-working task that is costly and takes time. Therefore, a requirement for our commit protocol is to be still compatible with 2PC, so existing and reliable concepts can still be used in mobile environments with an enhanced failure model.

1.1 Contributions

The main contributions of this paper are:

- we present a multiple coordinator atomic commit protocol for mobile networks that tolerates network partitioning as long as one partition contains a majority of coordinator votes
- the proposed protocol makes use of controlled failures; in such cases, a single coordinator is sufficient to complete the coordination
- we develop a mathematical model for the protocol failure probability and outline important criteria regarding the chosen number of coordinators
- we prove the correctness of the protocol
- we experimentally evaluate the amount of protocol overhead and the protocol failure probability.

Beyond our previous paper [2], in this contribution we

- prove protocol safety (correctness) and protocol liveness
- compute the protocol blocking probability
- experimentally determine its blocking probability
- experimentally evaluate the protocol performance and justify practical protocol usability
- give an important criterion for selecting an appropriate number of used coordinators.

The rest of our paper is organized as follows: Section 2 describes the requirements to MCP, the underlying assumptions, and the system architecture. Section 3 presents MCP, proves correctness, and analyses costs in terms of number of messages. Section 6 shows the experimental results, while Section 7 discusses related work. Finally, Section 8 concludes the paper.

*This work was supported in part by National Science Foundation Grant No. IIS-0312746

2 Architecture and Requirements

2.1 Requirements for Atomic Commit Protocols

Our atomic commit protocol design shall meet the following requirements. A transaction in our system consists of one or more sub-transactions running on individual participating databases. All transactions and their sub-transactions are considered to be non-compensatable.

Our protocol shall proceed correctly and reduce the blocking of the participating databases in case of network partitioning. In mobile networks, due to nodes' mobility and power limitation, databases and coordinators may fail or disappear at any time. Therefore, our protocol should be able to handle these disconnections and come to a one-sided commit decision, i.e. if one participant votes for abort, the decision must be abort. In addition, the protocol should proceed even if some coordinators or participating databases fail.

One may reasonably expect that the idea of using multiple coordinators [13] will reduce blocking. For the special case of a network partitioning where each partition contains at most half of all coordinators, [16] have shown that no non-blocking atomic commit protocol for asynchronous networks exists. However, in all other cases of network partitioning, we want to reduce blocking to a minimum extent.

In situations where it is possible to totally exclude network partitioning or to detect that no partitioning has occurred, our protocol should even be non-blocking as long as at least one coordinator still works.

2.2 Underlying Assumptions

If a database DB_i is unreachable or totally fails after it has sent its commit vote for a sub-transaction, we do not count this as a violation of the atomicity constraint. After the total failure of DB_i, it does not matter whether or not the database has executed the corresponding sub-transaction since it will not be available anymore. However, if DB_i reconnects to the network or recovers, DB_i must first commit or abort its sub-transaction, depending on the commit decision of the global transaction in which the sub-transaction participated, before DB_i is allowed to execute other transactions.

When a node detects that it is not able to complete its sub-transaction (e.g. due to a loss of power) or is going to disconnect from the network soon, the node tries to inform other nodes about this imminent failure. If it can inform at least one of the non-failing participants before the failure occurs, we call this a *controlled failure*.

We assume that the probability that a node will be able to complete the commit coordination of a given transaction is greater than the probability that it will fail or disconnect before a commit decision is made.

2.3 Architecture

We assume that in our ad-hoc network, we can identify some nodes that are likely more stable than others and closely connected to each other. To enhance availability of the coordination process, we use these nodes as a *cluster of coordinators.*. To do so and in contrast to traditional 2PC, we split the roles of the participants in such a way that we allow the initiator, the databases, and the coordinators running on different machines. Whenever an application starts a transaction involving more than one database, the node on which the application started the transaction becomes the Initiator which then sends the sub-transactions of the transaction to the corresponding databases and starts MCP, which runs on a group of n coordinator nodes that handle transaction commitment. Each coordinator in this cluster of coordinators is responsible for managing commit decisions for one or more databases. In the cluster of coordinators, there is one selected coordinator called the *main coordinator,* which makes global commitment for all coordinators in the cluster.

The databases' communication with a coordinator of this cluster of coordinators is similar to 2PC: Each participating database sends its vote for the transaction to a coordinator and receives the commit decision from this coordinator. However, the database might also ask other coordinators about the commit decision if some coordinators have failed.

After a certain time, the coordinators forward their collected votes to the main coordinator, which then makes the global commitment decision and spreads this decision to all coordinators using 3PC.

3 Multiple Coordinator Protocol

Our solution is based on the idea that the protocol failure probability may decrease when using multiple coordinators. To show this benefit, we use a failure model that is based on the *coordinator failure probability* p, indicating the probability of a single coordinator failure or disconnection during the coordination of *one* coordinated transaction. This singular failure probability applies to each coordinator node whenever we use a set of identical participants like sensor nodes or robot devices.

In order to guarantee the correctness of our solution in asynchronous networks, we need a majority, i.e. more than 50% of all coordinators in one partition, since we cannot determine whether participants have failed or moved to another partition (cf. [16]).

Since the protocol will not continue working if 50% or more of n previously selected coordinators have failed, we show that the availability of the protocol increases when using an odd number of 3 or more coordinators, each having a failure probability p with $p < 0.5$.

The *protocol availability* of MCP with n coordinators in environments where network partitioning can occur, i.e. the probability that a majority of the n coordinators is still working, is:

$$\mathsf{protAvail} = \sum_{k=0}^{\lfloor \frac{1}{2}n-0.5 \rfloor} \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}$$

while the *protocol blocking probability* is:
$\mathsf{protBlocking} = 1 - \mathsf{protAvail}.$

[11] showed that for failure probabilities $p > 0.5$ a single coordinator is more stable than using quorum based approaches. However, even with $p < 0.5$, the protocol availability when using multiple coordinators can be worse than when using a single coordinator, e.g. for $n = 10$ coordinators and $p = 0.47$, we get: $\mathsf{protAvail} = 0.45262$.

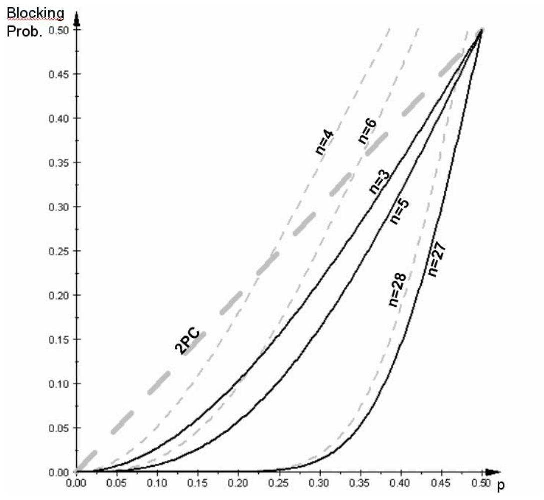

Figure 1. Calculated protocol failure probabilities for MCP having n coordinators

Figure 1 shows the blocking probability of multiple coordinators and of 2PC, which uses a single coordinator. On the x-axis, the failure probability of a single coordinator is shown, and each curve represents the blocking probability that occurs for MCP using n coordinators. The bold dashed line represents the blocking probability of a single coordinator which is the same as when using 2PC.

We can see from Figure 1 and the formula, that

1. the protocol availability is better when using an odd number of $n-1$ coordinators than when using an even number of n coordinators.
2. using an even number of coordinators each of which has a blocking probability of nearly 0.5 (e.g. 10 coordinators with $p = 0.47$) results in a worse blocking probability (i.e. 0.45) than when using a single coordinator (i.e. 0.47)
3. for an odd number of coordinators and $p < 0.5$, the blocking probability of MCP is always smaller than when using 2PC.

The reason for this result is that a cluster CE containing an even number n of coordinators tolerates as many coordinator failures as a cluster CO using the odd number $n-1$ of coordinators. This means, without tolerating more coordinator failures than CO, CE adds an additional source of failure. Therefore, instead of using an even number of coordinators, it is advisable to ignore one possible coordinator to get an odd number of coordinators, which implies a significantly lower blocking probability than that of 2PC. We therefore let MCP always use an odd number of coordinators.

To employ these multiple coordinators, we select a *cluster of coordinators*, which consists of nodes preferably in a single-hop distance to each other. The role of this cluster of coordinators is to come to a global coordination decision by using 3PC among coordinators and the main coordinator. For the communication of the databases and the cluster of coordinators, 2PC can be used. However, in case of a node failure within the cluster of coordinators, we need a termination protocol that guarantees a correct coordination decision.

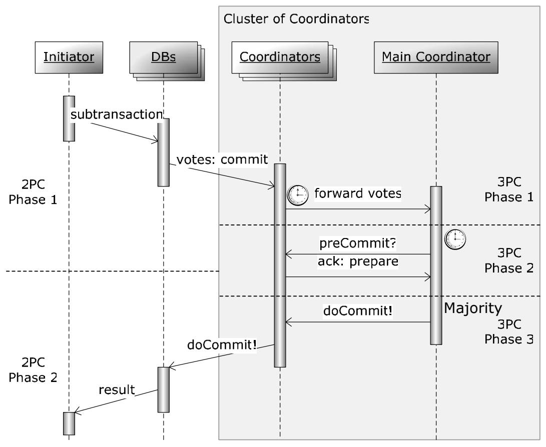

Figure 2. Sequence diagram (failure free case)

In cases where we cannot safely determine that the cluster of coordinators is not partitioned or where it seems that the main coordinator has failed, MCP runs a termination protocol which uses the ideas of the Paxos Consensus Protocol [9] to get a consensus decision on the transaction. This is described in Section 3.2. If we observe that network partitioning has not split the cluster of coordinators, the protocol can proceed without blocking the coordinators even if all but one of them have failed. In this case, the protocol availability is far better since only one remaining coordinator is sufficient.

Another advantage of our new protocol is that the cluster of coordinators is located in an environment with fast message transfer, e.g. in single-hop distance. This speeds up message transfer for the overhead caused by 3PC in the cluster of coordinators.

3.1 Failure Free Case

Since a detailed description of the protocol can be found in [2], we only describe the main ideas of MCP and concentrate on the correctness proof and experimental evaluation.

When there is no node or link failure, our protocol works as shown in Figure 2.

The protocol initiates execution by sending the subtransactions to appropriate participating databases including necessary coordination information (e.g. which is the main coordinator; which are the other coordinators).

Each participating database executes the sub-transaction until it can decide to vote for commit/abort and sends its vote to its associated coordinator. After the Initiator has selected the coordinators by different criteria and notified that they coordinate a transaction T, each coordinator including the main coordinator starts a local timer and accepts votes for a transaction T from the participating databases for a certain period of time. The timeouts that are used can be the same timeouts that are used when executing 2PC. When the time has passed, the coordinators will no longer accept votes from the participating databases on behalf of the transaction. The coordinators then bundle their collected votes and forward them to the main coordinator.

When the main coordinator has received all database votes or a specified period of time has passed, the main coordinator decides for commit of T if he received a commit vote from each database, and for abort otherwise. This decision is sent to the coordinators by using a modified version of 3PC: The main coordinator first sends a "prepare to send commit decision to databases" message to all coordinators. Each coordinator acknowledges the arrival of this message by sending a reply to the main coordinator. However, unlike 3PC, the main coordinator only needs a majority of coordinator acknowledgments to proceed. This means, after more than 50% of all coordinators have acknowledged the prepare message, the main coordinator sends a "forward commit decision to databases" message to the coordinators.

The coordinators follow this instruction and forward the commit decision to their associated databases, i.e. they perform the second phase of 2PC. The databases then commit or abort the transaction.

If the decision message is lost or delayed or if the corresponding coordinator disconnects, then the decision message does not reach each database. In this case, the databases themselves can also ask for the global decision. Finally, when a database has executed or aborted its subtransaction, it sends the result to the Initiator.

In summary, from the external point of view, the whole cluster of coordinators acts as one commit coordinator and the databases act as the participants in 2PC, while inside the cluster of coordinators the main coordinator acts as the commit coordinator and the individual coordinators act as the participants in a modified version of the 3PC protocol.

3.2 Failure Handling

We explain different possible failures that might occur during the execution of MCP and how our protocol handles these failures. A failure does not necessarily mean a complete crash of the participant, failures also include a disconnection of nodes from the network.

3.2.1 Controlled Failures

We define a *controlled failure* of the coordinator C as follows:

Definition 3.1: *A* controlled failure *of a coordinator c is a failure which can be detected in advance (e.g. due to low battery or machine shutdown) and the coordinator's commit status is known to at least one other non-failing coordinator.*

In contrast, an uncontrolled failure *of a coordinator c is a failure where the commit status information of c is lost, i.e. unknown to the non-failing coordinators.*

If a coordinator C_1 is able to determine a controlled failure of a different coordinator C_2, any informed coordinator can take over the vote and the execution state of the failed coordinator by starting a new, second instance of the coordination process on its own machine. To ensure progress, the coordinator that took over the instance must inform the cluster of coordinators about the old and the new address of the moved coordinator instance.

If a controlled failure occurs, i.e. the coordination process can be transferred to another node, this can be treated as when no failure occurred at all. For the simplification of our algorithm description, we assume that each coordinator node only executes a single process. This means, if a controlled failure occurs, we assume that the process is transferred to a new coordinator node that was not elected as a coordinator node before.

3.2.2 Coordinator Failure (except main coordinator)

Since MCP still works correctly if more than 50% of all coordinators work within the same network partition, a single coordinator failure has no influence on the protocol termination. If 50% or more of the coordinators fail, the main coordinator does not get a majority and must wait until sufficiently many of the failed coordinators have reconnected.

If a database does not get the decision message from the coordinator that should have sent the message, the database selects another coordinator from the cluster of coordinators. From the databases' point of view, the databases themselves are the active components and are asked to find a running coordinator.

3.2.3 Main Coordinator Failure

Each coordinator is allowed to suspect a main coordinator of failing. In this case, MCP uses two concepts, *version numbers* and *quorums*, which are also used in the Paxos Consensus [9] to terminate a transaction even in the case of a network partitioning where one partition contains a majority of all coordinators.

Version numbers are used to identify a unique main coordinator each time, while quorums enable a decision to become valid after a majority of coordinators have agreed on the decision in case of network partitioning.

Whenever a coordinator does not get an expected message from the main coordinator after a certain time, this coordinator is allowed to suspect a main coordinator failure and take over the main coordinator's role. We call this coordinator an *interim main coordinator.* Since every participating coordinator is allowed to decide that a main coordinator failure has occurred and to take over the main coordinator's role, there may be more than one main coordinator at a time.

To ensure having one interim main coordinator c_j that has a privileged position, each new main coordinator assigns itself a version number $v_{c_j}^{\text{new}}$ which is greater than the highest version number v of which the coordinator c_j has knowledge. To make sure $v_{c_j}^{\text{new}}$ is unique, each coordinator has a unique offset value o_{c_j}. The following formula shows how $v_{c_j}^{\text{new}}$ can be calculated in case of n coordinators: $v_{c_j}^{\text{new}} = \lceil v/n \rceil \cdot n + o_{c_j}$. Each coordinator may only *acknowledge* messages of that interim main coordinator having the highest version number v of which the coordinator has knowledge. This assures that if there are two or more interim main coordinators at the same time, we have an order of version numbers of interim main coordinators.

The new interim main coordinator first needs the last coordination states of more than 50% of all coordinators with which the transaction coordination has started. The interim main coordinator then inspects the proposal (commit or abort) that was proposed by the main coordinator with the highest version number and from now on proposes this proposal.

A proposal becomes a *valid decision* when a majority of coordinators has received and acknowledged this proposal.

Lemma 4.1 shows that if a proposal becomes a valid decision, it is ensured that another new main coordinator will get to know the latest decision. The reason is that every new main coordinator must also receive the latest states of more than 50% of all coordinators and creates a proposal that adopts the latest proposal. Therefore, a valid decision cannot be changed anymore even in case of network partitioning.

3.2.4 Network Partitioning

If network partitioning occurs, there can be at most one partition containing more than 50% of all coordinators that the transaction has started with. This partition continues protocol execution as in the case where the nodes that belong to other partitions have failed.

If at most 50% of all coordinators with which the transaction has started are within the same partition, these coordinators are blocked since any new interim main coordinator will receive at most 50% of the latest coordination states of all coordinators. This blocking is proven to be unavoidable, cf. [16].

4 Proof of Correctness

To prove the correctness of our algorithm, [8] pointed out that there are two fundamental properties which must be fulfilled: safety and liveness.

To ascertain safety, nothing wrong or undesirable may happen. To ascertain liveness, [1] states that the desired outcome must be reached sometime – in our case the main coordinator sends a decision that is valid and the coordinators forward the decision to the databases, each of which executes its sub-transaction.

To prove that these two properties hold, we first point out that a decision accepted by a majority cannot be changed anymore:

Lemma 4.1: *When a majority of coordinators has accepted a decision d with version number v from a main coordinator $m_{\texttt{success}}$, any new interim main coordinator $m_{\texttt{new}}$ can only propagate the decision $d_{\texttt{new}} = d$.*

Proof: *Any new main coordinator $m_{\texttt{new}}$ must first query a majority of coordinators for the latest proposal, before $m_{\texttt{new}}$ can adopt and propose this proposal itself. We order the coordinators that get a majority by their version number. Let $m_{\texttt{nextsuccess}}$ be the next coordinator after $m_{\texttt{success}}$ getting a majority and let $m_{\texttt{nextsuccess}}$ have the version number $v_{\texttt{nextsuccess}}$. Since $m_{\texttt{success}}$ was successful by assumption, a majority of coordinators know decision d. Therefore, at least one coordinator c_r will reply to $m_{\texttt{nextsuccess}}$ by sending the decision d and the version number v. As no coordinator with a version number between v and $v_{\texttt{nextsuccess}}$ got a majority, d will be the decision with the highest version number. Therefore, $m_{\texttt{nextsuccess}}$ will adopt and propose d. The same argumentation holds for all further coordinators $m_{\texttt{new}}$ that get a majority. Since only coordinators that get a majority can distribute a decision, all further decisions will be identical to d.* $\square$

Theorem 4.2: *Our termination protocol fulfills the property of safety.*

Proof: *According to [1], the safety conditions corresponding to our case which must be fulfilled are:*

- *Only a proposed value may be chosen. For our protocol, this means that only if all databases have voted for commit, a commit decision can be chosen.*
- *Exactly one decision must be chosen.*

The first condition is fulfilled because the initial main coordinator will only propose commit if it has received commit-messages from all databases. Any further main coordinator can only propose commit if a previous proposal was also commit. However, this can only happen if it was the initial main coordinator which proposed commit the first time. Therefore a commit decision implies that all databases have voted for commit.

The second condition is fulfilled since a valid decision can only be sent if more than half of the coordinators have acknowledged a proposal. Lemma 4.1 guarantees that this decision will not be changed. Therefore, after the execution of our algorithm, all running coordinators come to the same decision. $\square$

If we do not have different time intervals for a delayed restart of the distributed main coordinator selection algorithm, the following problem could occur: If two main coordinators I_1 and I_2 with version numbers $v_1 < v_2$ act at the same time, I_1 will be rejected by coordinators that already got in contact with I_2 due to the greater version number. If I_1 restarts its algorithm immediately with a higher version number v_1', it cancels the algorithm execution of main coordinator I_2. I_2, however, can start again with a higher version number and may cancel the algorithm execution of main coordinator I_1. This results in a loop.

In order to prevent this kind of loops, we increase the time a coordinator waits until it restarts its coordination efforts by a previously specified amount.

Theorem 4.3: *If no network partitioning lasts forever, our termination protocol fulfills the property of liveness.*

Proof: *To ascertain liveness, [10] asserts that the algorithm must terminate. To violate this criterion, every new main coordinator must be unsuccessful. If we have two active main coordinators at the same time, the coordinator with the higher version number is not stopped by the coordinator with the lower number. Since every coordinator C has a specified increasing time interval to wait before becoming a the newest interim main coordinator, there exists a number of rounds for which the corresponding waiting time has increased such that it is greater than the time that is needed to make a main coordinator's proposal valid. Therefore the algorithm terminates.* $\square$

5 Communication Overhead

5.1 Number of messages

We consider the number of messages in the normal case where a transaction is committed successfully. We assume that we have d databases and n coordinators, including the main coordinator. When the Initiator distributes the transaction to the participating databases, the necessary transaction information (e.g. which is the main coordinator; which are

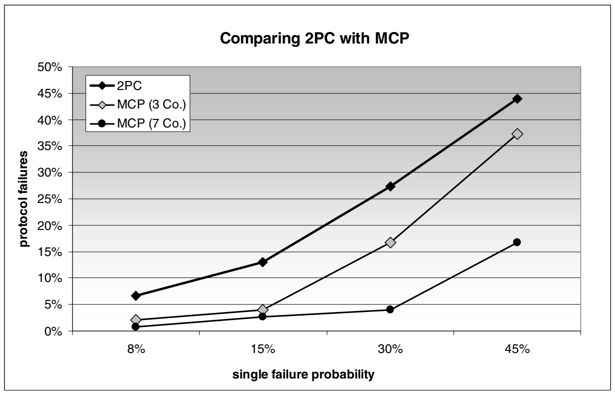

Figure 3. Comparison of 2PC, MCP (3 Coordinators) and MCP (7 Coordinators)

the other coordinators) is sent within the same message, i.e. it does not require an extra message. In the failure free case, we have, as shown in Figure 2, at most

- $d + d$ messages: Initiator to databases containing the sub-transaction, plus the reply to the coordinators containing the database's votes
- $n-1$ messages sent from $n-1$ coordinators to the main coordinator containing a summary of the databases' votes inside
- $3(n-1)$ messages using the 3PC
- $d+d$ messages: "do Commit" from the coordinators to the databases, and the reply from the databases to the Initiator.

Summing up all messages, we have a total of $(4d+4(n-1))$ messages. $4(n-1)$ messages are sent within the cluster of coordinators where message transfer is very fast due to one-hop communication among coordinators. The remaining messages are the costs of the normal 2-Phase-Commit which is implemented in many database applications.

5.2 Log Accesses

Since databases can use the same interface as within 2PC, the databases must write the same states and information into their log.

The coordinators, i.e. the participants of the cluster of coordinators have to log the final decision to inform the databases after a failure. However, they are not required to log every state of the 3PC execution since our protocol proceeds if more than half of the coordinators are running. Therefore, if a coordinator fails and returns, the coordinator can be either passive and acknowledge a running main coordinator, or it starts acting as an interim main coordinator. However, in both cases, any previous state that was not final is outdated. Therefore, we can omit any logging within the recovery process if no final decision was reached. When the final decision has been made, this information must be written to the log.

6 Experimental Results

6.1 Experimental Setting

We compare the stability of MCP with that of 2PC by simulating the use of both protocols within a mobile network having node disconnections and failures. We simulate database activity by delaying the transaction vote messages, and we simulate node disconnections and node failures by stopping the coordination process.

In order to compare MCP and 2PC, it is not necessary to simulate database failures for the following reasons. First, every database failure that occurs before sending the vote message implies an "abort" decision, which is a valid decision on the transaction and has no blocking effect on other databases. Second, database failures occurring after the vote message has been sent would not violate the atomicity constraint as described in Section 2.2. Therefore, we have omitted the simulation of these kinds of failures in our experiments.

Although the handling of node movement is a task of the routing layer, we simulate those cases where node movement leads to disconnections, since in such cases nodes cannot coordinate anymore. Any other case of node movement will only lead to an increased message delivery time and – depending on the timeout values – to a transaction abort, but any simple node movement without disconnections will not result in a protocol failure.

In our simulation, each coordinator runs a random number generator after it is chosen as coordinator of a transaction: the generated random number determines *whether* the coordinator fails (with the probability p) and, should it fail, *when* it fails (this probability is uniformly distributed within a maximum time window). This favors a fast protocol, since the simulated coordinator failure may also occur after a transaction was successfully coordinated; in such case the failure has no effect on the fast protocol.

We simulated the coordination of 150 transactions and measured the number of *protocol failures* for various single node failure probabilities p. We assumed that a protocol failure occurred if the protocol is not able to make a commit decision and inform the databases and the initiator[1] about this decision within a maximum time limit (i.e. 30 seconds). Additionally, we measured the time from the initialization of the protocol until the notification of the initiator. The database activity time for each transaction is equal for 2PC and MCP.

Within MCP, we used the following timeouts:

`MainCoordinatorDecision = 5s` — The time, after which the main coord. makes a commit decision

`MainCoordinatorFailureDetection = 10s` — The time, after which the first coordinator may decide that the main coordinator has failed and take over the main coordinator's role.

`CoordinatorForward = 3.2s` — Indicates the time, after which each coordinator forwards its collected databases' votes.

`DatabaseActivity = 0s - 3s` — Indicates in both MCP and 2PC the time that the database uses for performing the transaction, i.e. the time that a database waits before sending the vote message

6.2 Protocol Failures

Figure 3 compares the percentage of failed coordinations for both 2PC and MCP (using 3 and 7 coordinators). On the

[1]in our simulation, the initiator never fails since an initiator failure has no influence on the protocol availability

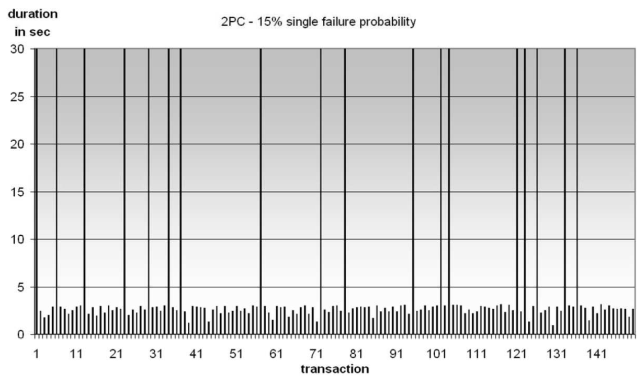

Figure 4. Transaction duration for 2PC

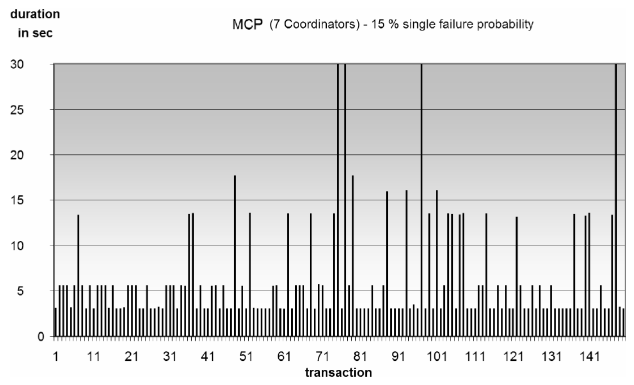

Figure 5. Transaction duration for MCP

x-axis, the single failure probability p of each coordinator is shown. For example, a single failure probability of 15% means that each participating coordinator will not finish a transaction coordination with the probability of 0.15

On the y-axis, the amount of protocol failures is shown. The amount of 2PC protocol failures does not exactly correspond to the simulated single coordinator failure probability for the following reason: since each coordinator that is supposed to fail does not fail immediately but within a time window, a simulated coordinator failure might also occur after a fast transaction has been finished, and therefore this simulated coordinator failure does not lead to a protocol failure.

It can be seen by the curves indicating the failure probabilities for MCP that using MCP assures a significantly lower failure rate than 2PC. In addition, we can conclude that using more coordinators, e.g. 7 coordinators, is especially appropriate within environments where the single failure possibility of a device is in the range of 15% to 45%.

6.3 Transaction Duration

Although we have just seen that MCP has fewer protocol failures than 2PC, Section 5 states that MCP needs $4 \cdot n$ more messages than 2PC. However, the number of messages is not the only interesting parameter, e.g., messages between databases and the cluster of coordinators take more time than messages within the cluster of coordinators. To show that MCP can be used practically, we need to measure the amount of time that is required to complete a transaction. For this reason, we simulated database activity for each transaction by delaying the vote message, and we measured how long it took until all databases got the commit decision. If the commit decision did not reach one or more databases within a defined time limit (i.e. 30 seconds), we counted this missing coordination decision as a protocol failure and we set the transaction duration to the defined time limit.

Figure 4 and Figure 5 consist of bars, each showing the time (in seconds) that was needed to complete the corresponding transaction with a single failure probability of 15%. Bars exceeding 30 seconds indicate a protocol failure for the corresponding transaction.

The effects of the termination protocol that is used by MCP can especially be seen in Figure 5. Bars with y-axis values around 3 seconds show transactions where none or one coordinator have failed and all vote message have been sent to the main coordinator. Bars around 5s indicate that a `MainCoordinatorDecision` timeout has occurred, i.e. a coordinator failed before sending the commit vote.

Higher bars occur if the main coordinator or participants have failed. In this case, the coordinators detect this failure after the `MainCoordinatorFailureDetection` timeout and elect a new main coordinator that completes the transaction.

When we compare the average transaction coordination times of 2PC and MCP, we can not only see that the number of protocol failures of MCP is much lower than that of 2PC, but also that in the cases where no coordinator fails, the time needed for MCP is slightly greater than that of 2PC, but shows less jitter. The explanation for this result is that MCP uses two timeout values: one for the coordinator, the other for the main coordinator. Since a coordinator forwards its received database votes only after the time of the coordinator timeout has passed, the main coordinator receives the vote messages almost at the same time. The single coordinator of 2PC, in contrast, may decide on the transaction's commit status after it has received all database votes.

If a coordinator failure occurs, MCP needs to run its termination protocol, which takes additional time. This additional time is caused by the (adjustable) timeout that the termination protocol must wait before declaring a node as having failed. However, MCP ensures more successfully coordinated transactions, in contrast to 2PC, which was fast but often unsuccessful as can be seen by the number of bars exceeding 30 seconds.

How much MCP can improve the mean duration time of a transaction in comparison to 2PC depends on the number of coordinators used, their failure probability, and how we weight infinite blocking of non-decided transactions. When we weight the duration of a non-decided transaction as lasting 30 seconds, we can see that the average transaction time for single failure probabilities of 15% in MCP with 7 coordinators is on average 30% faster than 2PC.

7 Related Work

Atomic commit protocols for mobile networks have to solve three main problems: message loss, node failure, and network partitioning.

Under the assumption of message loss, [4] even proved that a commit decision is not possible within the coordinated attack scenario, in which the commit decision is that two generals use an unreliable communication channel to agree on a time for a common attack. However, our requirements do not include a time-limit for the commit decision unlike the coordinated attack scenario. Therefore, we can guarantee an atomic execution unless all coordinators fail.

In contrast to our previous work [2], in which the idea of using a combination of 2PC [4], 3PC [14], and Paxos Consensus [9] was first described, our present paper theoretically examines the protocol blocking probability and shows important criteria for a good selection of multiple coordinators (i.e. usage of an odd number of coordinators; usage of multiple coordinators only if $p < 0.5$). In addition, we have verified the applicability of these theoretical results within practical experiments and have gotten experimental results not only for the number of protocol failures for different values of p, but also for the time that is needed by 2PC and MCP for transaction coordination.

The use of more than one coordinator is also proposed in [13]. This approach attaches "backup coordinators" that store an already reached commit decision of a single commit coordinator. However, if the single coordinator fails before a commit decision is reached, the protocol blocks. MCP, in contrast, can detect a main coordinator failure and allows the coordinators to take over the main coordinator's role, which ensures that the protocol continues.

The proposed termination protocol is based on the idea of quorums, which was introduced in [3] and later applied to 3PC by [15]. However, since the latter protocol is blocking in case of cascading failures, we additionally use version numbers, like in Paxos Consensus ([9] and [12]) or E3PC [6], for identifying a version number order of commit coordinators. However, we do not follow [9] by using the Paxos Commit Protocol [5] to get a consensus on the commit state as this approach does not take advantage of one-hop-environments and has a significant message overhead when there are many coordinators. Furthermore, if the coordinators know that there is no network partitioning in the cluster of coordinators, our protocol is able to be non-blocking even with only one remaining coordinator (the above mentioned quorum protocols need at least ($\lfloor \frac{1}{2}f + 1 \rfloor$) remaining coordinators, where f denotes the total number of coordinators. If we cannot exclude or safely detect a network partitioning, our protocol also needs ($\lfloor \frac{1}{2}f + 1 \rfloor$) coordinators to be running in the same network partition to get a commit decision for the transaction. Otherwise, [16] proved that it is inevitable to wait until the network is connected again.

[7] suggests a different approach, which uses a timeout based proposal. However, this protocol assumes that transaction compensation is possible in case of participant failure. In contrast, we do not rely on this compensation assumption, since in case of network partitioning, this requirement cannot be fulfilled if compensation transactions will not reach devices that remain within different partitions. Our approach is more general in that it considers all transactions and their sub-transactions to be non-compensatable.

8 Summary and Conclusion

We have shown that traditional atomic commit protocols for fixed-wired networks are not suitable for mobile ad-hoc networks due to their blocking behavior in case of node or link failures, message loss, and network partitioning.

Furthermore, we have shown that using multiple coordinators needs some care and is not always better than using a single coordinator: First, an odd number of coordinators should be chosen, second, if the single coordinator failure probability is greater than 0.5, only a single coordinator should be chosen. However, our experiments have demonstrated that using multiple coordinators obeying these coordinator selection rules results in a much higher protocol availability.

Especially for the use in mobile ad-hoc networks, we have developed MCP, a multiple coordinator atomic commit protocol that takes advantage of special mobile network structures like one-hop communication. Since MCP's logic resides in the cluster of coordinators and databases communicate with the cluster of coordinators by using 2PC, MCP can be easily implemented in database systems that already use 2PC as only slight modifications to 2PC are needed. We have shown the correctness of our protocol and experimentally evaluated MCP regarding the number of protocol failures and protocol speed. It turned out that MCP shows fewer protocol failures than 2PC and is practically usable regarding the protocol speed.

Therefore, we consider MCP a useful contribution for transferring fixed wired applications to mobile environments.

References

[1] B. Alpern and F. B. Schneider. Recognizing safety and liveness. Technical report, Ithaca, NY, USA, 1986.

[2] J.-H. Böse, S. Böttcher, L. Gruenwald, S. Obermeier, H. Schweppe, and T. Steenweg. An integrated commit protocol for mobile network databases. In *9th International Database Engineering & Application Symposium IDEAS*, Montreal, Canada, 2005.

[3] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, pages 150–162, 1979.

[4] J. Gray. Notes on data base operating systems. In *Advanced Course: Operating Systems*, pages 393–481, 1978.

[5] J. Gray and L. Lamport. Consensus on transaction commit. *Microsoft Research – Technical Report 2003 (MSR-TR-2003-96)*, cs.DC/0408036, 2004.

[6] I. Keidar and D. Dolev. Increasing the resilience of atomic commit at no additional cost. In *Symposium on Principles of Database Systems*, pages 245–254, 1995.

[7] V. Kumar, N. Prabhu, M. H. Dunham, and A. Y. Seydim. Tcot-a timeout-based mobile transaction commitment protocol. *IEEE Trans. Comput.*, 51(10):1212–1218, 2002.

[8] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

[9] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[10] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982.

[11] D. Peleg and A. Wool. The availability of quorum systems. *Information and Computation*, 123(2):210–223, 1995.

[12] R. D. Prisco, B. W. Lampson, and N. A. Lynch. Revisiting the paxos algorithm. In *Distributed Algorithms, 11th International Workshop, WDAG '97, Saarbrücken, Germany*, Lecture Notes in Computer Science, pages 111–125. Springer, 1997.

[13] P. K. Reddy and M. Kitsuregawa. Reducing the blocking in two-phase commit with backup sites. *Inf. Process. Lett.*, 86(1):39–47, 2003.

[14] D. Skeen. Nonblocking commit protocols. In Y. E. Lien, editor, *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan*, pages 133–142. ACM Press, 1981.

[15] D. Skeen. A quorum-based commit protocol. In *Berkeley Workshop*, pages 69–80, 1982.

[16] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. In *Berkeley Workshop*, pages 129–142, 1981.