ABSTRACT
The problem of blocking in two-phase commit protocols (2PC) is one of the main issues to solve when designing a reliable, available and efficient distributed system. To resolve the blocking problem three-phase commit protocol (3PC) was introduced using an extra round of message transmission, which is not an ideal solution. Instead it is desirable to keep the number of message exchanges on a low level and for that reason a commit protocol should be efficient in keeping the number of phases as minimal as possible, which redirect us to the two-phase commit protocol. In order to achieve the non-blocking property there have been presented a variety of solutions throughout the years, and in this paper we have chosen to focus on two different approaches: one with a protocol that introduce the usage of backup sites (BC) and another that intends to implement a non-blocking two-phase commit protocol (N2PC) with a cooperative recovery. To maintain a low level of message exchanges the first protocol does not use acknowledgements, as a substitute the participants handles that responsibility and to obtain a non-blocking property, election of a new coordinator. Storing copies of coordinator decisions on a backup site is the foundation of the second algorithm.

Keywords: coordinator, participant, commit protocol, non-blocking.

1. INTRODUCTION
How to improve two phase commit protocols (2PC) with non-blocking features? 2PC is an atomic blocking protocol and when non-blocking features are desired a three phase commit protocol (3PC) is normally used. 3PC has a much greater overhead cost than 2PC and this paper shows how to benefit from both kinds of protocols. For a brief introduction to 2PC protocols, see [1].

Atomic commit protocols are used in distributed systems when several sites need to update their databases with the same information. One client requests information to be uploaded and a site receive the request and start a procedure where he becomes the coordinator of this request. The other sites in the system will then become participants of the particular request.

The atomicity property of the protocol means that the transaction must be performed at all sites or not at all; this is achieved by letting all participants vote YES or NO to the particular transaction depending if they can commit it or not. Only when all sites are ready to commit, the coordinator sends a GLOBAL_COMMIT to the participants as confirmation that they can commit the transaction. A site can be both coordinator and participant at the same time but for different transactions.

If a coordinator site crashes and a participant(s) waits for a final answer from the coordinator, if he should commit the transaction or not, the participant(s) is blocked as long as the coordinator is down. This is the problem with the 2PC algorithm, while 3PC have more communication between the coordinator and the participants, and can therefore avoid this problem. Though, all communication takes valuable time in a distributed system where the sites can be far away from each other, a 2PC-protocol is to prefer. Table 1 shows comparison between 2PC and 3PC regarding message exchanges, log writes and degree of blocking where n is the number of participants; this is based on table 1 in [2].

Table 1. Comparison of commit protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Message exchange</th>
<th>Log writes</th>
<th>Degree of blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic 2PC</td>
<td>4(n-1)</td>
<td>2n</td>
<td>high</td>
</tr>
<tr>
<td>3PC</td>
<td>5(n-1)</td>
<td>2n</td>
<td>low</td>
</tr>
</tbody>
</table>

It is the extra phase in 3PC can gives the extra n-1 message exchanges compared to 2PC. If the distributed system has a lot of transactions to be executed, this will become a significant performance loss.

We have investigated two variants of non-blocking techniques. One which uses backup sites (BC) that store all decisions from the coordinator in case he goes down, then a participant can reach the final decision at the backup site. In this variant, there is still a slight possibility of blocking if both the coordinator and backup site are down at the same moment, but if the backup sites are selected with respect to failure independence; this is very rare. If a participant after all should be blocked, this protocol has the benefit that the participant only waits for the first site to recover. This technique is described in paper [3]. The other technique avoids unnecessary transaction aborts (N2PC) and is described in paper [2], this algorithm assumes that the coordinator always votes commit and that all participants are able to communicate with each other. A coordinator makes a list with all available sites and this list is sent together with the VOTE message to all participants. In this way when a coordinator fails, the participants know which site that takes the role as coordinator and continue with the transaction. When the original coordinator comes up after a failure it becomes a participant of the transaction.

This paper is organized as follows. In section 2, the commit protocols 2PC and 3PC are described. In section 3, we describe...
two algorithms that improve 2PC with non-blocking feature. Our conclusions come in section 4.

2. COMMIT PROTOCOLS

2.1 Two-Phase Commit Protocol

The two-phase commit protocol is one of the most common ones regarding atomic communication in distributed systems. That is because it is quite simple and straightforward. There exist several variants of the 2PC algorithm and the described algorithm further down are a combination from [6] and [7]. The algorithm works as all sites must accept a transaction before it is executed, if only one site votes no to the transaction then it is aborted for all sites.

2.1.1 Phases

The 2PC algorithm goes through two phases: the Voting phase and the Commit phase. The voting messages from a participant to a coordinator are YES or NO depending of the decision at the participant whether to vote yes or no to the requested transaction. The commit messages from a coordinator to the participant are GLOBAL_COMMIT or GLOBAL_ABORT depending if all participants have voted yes or not. All decisions, at each site, are logged in their respective write-ahead-log along with the transaction. The write-ahead-log must be in a stable storage to ensure that data is not lost during a site failure.

Voting phase:
1. Coordinator sends a VOTE_REQUEST along with the transaction to all participants.
2. Each participant receives the VOTE_REQUEST.
   a) If the participant can commit the transaction a YES is sent to the coordinator.
   b) If the participant cannot commit the transaction a NO is sent to the coordinator.

Commit phase:
3. Coordinator waits for replies from all participants.
   a) If all participants have answered YES, a GLOBAL_COMMIT is sent to the participants.
   b) If at least one participant has voted NO, a GLOBAL_ABORT is sent to the participants that have voted YES. The ones which have voted NO already know the decision.
4. Participant waits for reply from coordinator.
   a) If a GLOBAL_COMMIT message is received, the participant commits the transaction and sends an ACK (acknowledgement) to the coordinator.
   b) If a GLOBAL_ABORT message is received, the participant aborts the transaction.

2.1.2 States

The 2PC algorithms go through different states depending on the decisions from the coordinator and participants. In figure 1 and 2, we show the finite state machines (FSM) for the coordinator and a participant, with the numbers from the phases in section 2.1.1 showing the transitions between the states.

In the WAIT state for the coordinator, when the coordinator waits for replies from all participants, we are not certain if the next state will become ABORT or COMMIT. If a timeout occurs the decision will become GLOBAL_ABORT, see section 2.1.4.

The READY state for the participant is in some literature named the unsure state, because when a participant is between the moments he has voted COMMIT and waiting for a GLOBAL_ABORT or a GLOBAL_COMMIT from the coordinator. In this state the participant does not know whether the final decision will be a GLOBAL_COMMIT or GLOBAL_ABORT. It is in this state the 2PC protocol can be blocked, when the coordinator is down. The participant tries to handle the situation with timeout and talk to other participants; this is described in section 2.1.4.

2.1.3 Atomic commitment

Atomic commitment in 2PC is an essential demand for the algorithm. It means that every transaction must be handled the same way at all sites, either commit the transaction or abort it. To achieve atomic commitment we need to fulfill the following conditions according to [5].

A. Every participant and the coordinator must reach the same decision
B. A GLOBAL_COMMIT will be reached only if all participants and the coordinator have voted commit.
C. When every participant and the coordinator votes commit and there are no failures the result must be GLOBAL_COMMIT.

D. A reached decision by a participant or the coordinator is not reversible.

E. If tolerated failures occur and get repaired in a reasonable short time, then all participants and coordinator finally will reach a decision.

These conditions will be further discussed in sections 2.1.4 and 2.1.5.

2.1.4 Timeouts

To deal with failures in the communication or if a site fails and gets unreachable temporarily, the 2PC algorithm uses timeouts for recovery to avoid that the other sites have to wait too long.

Thus there are only three occasions when a site is waiting for a message, and that is in states INIT (2), WAIT (3) and READY (4).

In the participants state INIT (2), he waits for a transaction request from the coordinator – before the participant votes - and as any site is able to solely decide to abort the transaction before it has voted commit, if a participant times out in this stage, he can just vote NO and stop.

In the coordinators state WAIT (3), he is waiting for answers from the participants on a vote request. Here he can choose to send GLOBAL_ABORT if a timeout occurs as he has not reached a decision yet.

Considering the participants state READY (4), a participant that has voted commit is in the unsure state, and does not know if it will commit or abort the transaction if a timeout occurs. Based on these presumptions, contrasting to the other two cases, the participant is deemed to consult another site, finding out what to decide. Supposing the participant cannot reach another participant or the coordinator he will remain blocked until he could. This presupposes of course that the participants not only knows about the coordinator but also is aware of the other participants, but that is easily achieved. This does not eliminate the blocking as all participants can be in the same situation, in which they are all waiting for a decision from the coordinator.

2.1.5 Recovery

When a site needs to update its data, due to missed transactions during its absence, it must undergo a recovery procedure. In this paper we only do it from a transactions point of view, in reality this can be much more complicated with several missed transactions: that case is not covered in this paper.

When a single site are about to recover from failure it uses the write-ahead-log to determine in which state it was when the failure occurred, and also what decision is to be decided. This is no big problem except for the case when a failure happens during the unsure state, which you will see after the following explanation:

While a participant \( n \) are recovering from a failure, to be able to fulfill the atomic commitment condition \( E \) from 2.1.3, \( n \) must arrive at the same decision as the other sites concerning the actual transaction. As participant \( n \) remembers the state he was in, through the write-ahead-log, at the moment he failed out. If he received a GLOBAL_COMMIT or GLOBAL_ABORT from the coordinator just before he went down, and also after unilaterally decided NO then he has already reached a decision. In these cases, \( n \) are able to recover independently, but that is not the case if \( n \) went down during the unsure state (state READY at the participant). If \( n \) has voted commit, he cannot know if the decision are GLOBAL_COMMIT or GLOBAL_ABORT and must therefore consult other sites to get the decision.

2.2 Three-Phase Commit Protocol

The three-phase commit protocol is a non-blocking protocol that is based on the two-phase commit protocol.

2.2.1 Phases

The 3PC protocol has one extra phase, called the pre-commit phase, compared to 2PC. It is this phase that makes this protocol non-blocking but it comes with the extra cost of message transfers as we see in the phase description below. As in 2PC, all decisions are logged at each site in their respective write-ahead-log to ensure recovery in case of site failure.

Voting phase:

1. Coordinator sends a VOTE_REQUEST along with the transaction to all participants.

2. Each participant receives the VOTE_REQUEST.
   a) If the participant can commit the transaction a YES is sent to the coordinator.
   b) If the participant cannot commit the transaction a NO is sent to the coordinator.

Pre-commit phase:

3. Coordinator waits for replies from all participants.
   a) If all participants have answered YES, a PREPARE_COMMIT is sent to the participants.
   b) If at least one participant has voted NO, a GLOBAL_ABORT is sent to the participants that have voted YES. The ones which have voted NO already know the decision.

4. Participant waits for reply from coordinator.
   a) If a PREPARE_COMMIT message is received from the coordinator, the participants commit the transaction and send an READY_COMMIT (acknowledgement) to the coordinator.
   b) If a GLOBAL_ABORT message is received from the coordinator, the participants abort the transaction.

Commit phase:

5. Coordinator waits for replies from all participants.
   a) If all participants answer READY_COMMIT, a GLOBAL_COMMIT is sent to the participants.
   b) If the coordinator gets a timeout waiting for a READY_COMMIT message from a participant, the coordinator knows that the participant has voted YES and now is unavailable. The coordinator sends a GLOBAL_COMMIT to the remaining participants.
6. When the participants receive the GLOBAL_COMMIT, they commit the transaction and send an ACK (acknowledgement) to the coordinator.

If the coordinator site goes down and a participant has not received GLOBAL_COMMIT, the participant checks with other participants and if they have received either GLOBAL_COMMIT or GLOBAL_ABORT it can safely accept the same. And if all participants have received PRE_COMMIT, they can all commit the transaction.

2.2.2 States
The 3PC algorithms go through different states depending on the decisions from the coordinator and participants. Figure 3 and 4 shows the finite state machines (FSM) for the coordinator and a participant, with the numbers from the phases in section 2.2.1 showing the transitions between the states. The difference compared to 2PC is the state PRECOMMIT in which the coordinator and participant are aware that the decision will be GLOBAL_COMMIT.

![Figure 3. Finite state machine for the coordinator](image1)

When a participant timeouts in state READY, he have to contact the other participants to find out which decision they have received. If another participant is in state COMMIT or ABORT, he moves to that state as well. If all participants are in state PRECOMMIT, they can all commit the transaction. This is the difference compared to 2PC protocol, where the participant cannot reach a decision.

3. NON-BLOCKING ALGORITHMS
In this paper we show two different kinds of non-blocking 2PC protocols; this should not be mistaken for as these non-blocking techniques are better than others, they are just examples. The first is from paper [2] and this protocol elects a new coordinator if the existing one fails. The second is from paper [3] and uses backup sites which are notified of the coordinators decision; this algorithm does not satisfy the non-blocking property to 100 per cent.

3.1 N2PC
The algorithm we will describe is the one proposed by Byun and Moon [5] and is a non-blocking 2PC (N2PC) protocol containing concurrent recovery and the basic concepts are as follows.

```
[START]
if ((site, exist in failure list)
  Reject T
else if (conflicts with schedule)
  Reject T
else
  WriteLog(YES, T)
  Broadcast(VOTE_REQUEST, T)
[YES]
if (all YES)
  WriteLog(GLOBAL_COMMIT, T)
  Commit
  Broadcast(GLOBAL_COMMIT, T)
[NO]
  WriteLog(GLOBAL_ABORT, T)
  Abort(T)
  Broadcast(GLOBAL_ABORT, T)
[TO]
  WriteLog(GLOBAL_ABORT, T)
  Broadcast(GLOBAL_ABORT, T)
  WriteFailureList(sitei)
[WHO]
  Send(CORD, coordinator)
[INQ]
  LookUp(T)
  if (committed) Send(GLOBAL_COMMIT, T)
  else Send(ABORT, T)
  DeleteFailureList(sitei)
```

![Figure 5. N2PC Coordinator Algorithm](image2)
First the coordinator only sends a transaction request if he himself will commit and therefore the participants are able to commit the transaction even if the coordinator fails. The election of a new coordinator if the old one does not recover fast enough, is the other criteria this algorithm are based upon, to avoid blocking.

There are some simplifying assumptions: both commit and abort are not reversible, all sites can communicate with each other and the network is reliable enough to not get split in halves (partitioning). We also assume that the system can detect a site failure by timeouts (TO) and introducing a set of messages (M) consisting of the following:

START, VOTE_REQUEST, YES, NO, TO, WHO, CORD, YOU, INQ, GLOBAL_COMMIT, and GLOBAL_ABORT.

START initiates a transaction, VOTE_REQUEST is for collecting decisions from all participants and detached is the list of the participants, YES and NO are the votes from a participant, GLOBAL_COMMIT and GLOBAL_ABORT are the final decisions from the coordinator, TO is the timeout as we have mentioned, WHO is for sorting out which site is the coordinator, YOU is for selecting a new coordinator based on the VOTE_REQUEST attached list, and INQ gives the status of the current transaction when the site failed.

The algorithm is divided in three parts, a coordinator part (figure 5), a participant part (figure 6.) and the last one for the recovery (figure 7). The algorithms below are from [2] with slight modifications of the message names.

The characterization of N2PC is the migration of the coordinator if there are at least two sites in a distributed system to supply continuous processing of transactions. Due to lack of scheduling or other control mechanisms the inquiring is required, even though there are no failures present at the moment. Because of this the coordinator first look-up if there are any process scheduling or faulty sites before he starts any distributed transaction. If a coordinator fails during a transaction a participant is elected as the new coordinator by receiving a YOU-message and then takes over the decision-making. At the moment an abdicated coordinator revives, he will take the role as a participant, passing over all his information including non-terminated transactions to the successor.

Here is the recovery process algorithm:

```plaintext
[REVIVE]
if (T in the log)
  Send(WHO)
[TO]
  Send(WHO, sitex, site)
if (failure list is NULL) blocked
[COORD]
  for (each T in log)
    Send(INQ, site, T)
[COMMIT]
  WriteLog(COMMIT, T)
  Commit
[ABORT]
  WriteLog(ABORT, T)
  Abort(T)
[TO]
  Send(YOU, T, site
[YOU]
if (all)
  WriteLog(COMMIT, T)
  Commit(T)
  Broadcast(COMMIT, T)
else
  WriteLog(ABORT, T)
  Abort(T)
  Broadcast(ABORT, T)
[WHO]
  Send(CORD, coordinator)
[START]
  Send(START, T) to the coordinator
```

Figure 7. Recovery process algorithm

3.1.1 Performance

Byun and Moon presents proof of the correctness of the N2PC algorithm in their article [2], that falls out of the scope of this paper, but some comparing due to performance with different kinds of 2PC protocols would be appropriate. For this purpose we use the basic two-phase protocol (B2PC), presumed commit (PC2PC) and presumed abort (PA2PC) protocols. PA2PC and PC2PC protocols reduce the ACK messages from the participants to the coordinator as the protocols assume GLOBAL_COMMIT or GLOBAL_ABORT respectively. These protocols are well known and therefore we do not describe them in detail, see [2] or similar for further details.

The 3PC will be excluded in this comparison due to the fact that it has one extra phase, which is a too big disadvantage here. Comparison of the performance will be done according to message exchange and log writes.

E lecting used in N2PC is more efficient then the periodic inquiring of the presumed protocols (PA2PC and PC2PC),
The backup site as well. The states is not quite the same and of course there is a FSM for 2PC, described in section 2.2.2, although the transitions between information. This algorithm has the same states in the FSM as the 2PC, and because of the schedule conflict checking which reduces the message exchanges. If there is no transaction abort PA2PC is equal to B2PC, and PC2PC is equal to N2PC. However, because it is impossible that the probability is zero, PA2PC is more efficient than PC2PC. And N2PC is the most efficient variant of these 2PC protocols, since there are no acknowledgements.

**Table 2. Message exchanges without site failures**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Phase</th>
<th>Voting</th>
<th>Local decision</th>
<th>Decision distribution</th>
<th>Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2PC</td>
<td>n-1</td>
<td>n-1</td>
<td>(1-P_a)(n-1)</td>
<td>n-1</td>
<td></td>
</tr>
<tr>
<td>PA2PC</td>
<td>n-1</td>
<td>n-1</td>
<td>(1-P_a)(n-1)</td>
<td>(1-P_a)^2(n-1)</td>
<td></td>
</tr>
<tr>
<td>PC2PC</td>
<td>n-1</td>
<td>n-1</td>
<td>(1-P_a)(n-1)</td>
<td>1-(1-P_a)^2(n-1)</td>
<td></td>
</tr>
<tr>
<td>N2PC</td>
<td>(1-P_a)(n-1)</td>
<td>(1-P_a)(n-1)</td>
<td>(1-P_a)^2(n-1)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The columns in table 2: Voting, Local decision, Decision distribution and Ack are corresponding to the messages VOTE_REQUEST, YES/NO, GLOBAL_COMMIT/ABORT, and ACK respectively.

### 3.2 BC

This 2PC algorithm has backup sites, one for each coordinator that is selected with respect to failure independence. That is, if for instance the coordinator fails, it does not affect a failure at the backup site. They do not share resources that might cause failure at both sites at the same time. This is an important criterion; otherwise this algorithm would be more or less useless.

#### 3.2.1 Phases

The BC algorithm goes through three phases: the **Voting phase**, the **Backup phase**, and the **Commit phase**. The original commit phase is divided in two parts, where the backup site first gets the decision and stores it before the participants receives the information. This algorithm has the same states in the FSM as the 2PC, described in section 2.2.2, although the transitions between the states is not quite the same and of course there is a FSM for the backup site as well.

We have rewritten the phase and message names from [3] due to easier comparison with the described 2PC protocol in section 2.1. All decisions, at each site, are logged in their respective write-ahead-log. Along with the transaction, the write-ahead-log must be in a stable storage to ensure that data is not lost during a site failure.

### Voting phase:

1. Coordinator sends a VOTE_REQUEST along with the transaction to all participants.
2. Each participant receives the VOTE_REQUEST.
   a) If the participants can commit the transaction a YES is sent to the coordinator.
   b) If the participant cannot commit the transaction a NO is sent to the coordinator.

### Backup phase:

3. Coordinator waits for replies from all participants.
   a) If all participants have answered YES, a DECIDED_TO_COMMIT is sent to the backup site.
   b) If at least one participant has voted NO, a GLOBAL_ABORT is sent to the participants that have voted YES.
4. If backup site receives DECIDED_TO_COMMIT from coordinator, it sends back RECORDED_COMMIT to confirm that he has stored the message in stable storage.

### Commit phase:

5. Coordinator receives RECORDED_COMMIT from the backup site and sends GLOBAL_COMMIT to all participants.
   a) If a GLOBAL_COMMIT message is received, the participant commits the transaction and sends an ACK (acknowledgement) to the coordinator.
   b) If a GLOBAL_ABORT message is received, the participant aborts the transaction.

These phases is similar to the phases in the 3PC protocol but in this case we only transmit extra messages to one site (backup site), and thereby reduce the message passing to a minimum and get non-blocking properties.

#### 3.2.2 Blocking

The blocking situation in 2PC, when a failure occurs at the coordinator and a participant is in state READY waiting for a decision. Then if a timeout occurs, the participant checks with the backup site to receive the decision. In this algorithm the backup site does not take over the role as a coordinator, instead when the coordinator goes up again it continues as a coordinator. A blocking can occur in the backup phase when the coordinator has sent DECIDED_TO_COMMIT to the backup site, and waits for a RECORDED_COMMIT. If the backup site is down in this moment we get a blocking. This situation must be handled, and Reddy and Kitsuregawa [3] propose following.

1. The coordinator waits until the backup site is up.
2. Another backup site is chosen and the information is sent to all sites.
3. If a backup site is not possible to chose, this information is sent to all sites and the algorithm continues as a regular 2PC algorithm with the risk of blocking.
As this paper aims at non-blocking 2PC algorithms, propose number 1 is not acceptable. Although, this proposal has the benefit that a participant only waits for the first site to recover (coordinator or backup site). But in general, we do not want to remove blocking in one part of the algorithm and get a new blocking possibility at another part. Alternative 3 should only be possible as a last resort. We propose alternative 2, where a new backup site is chosen. Although this will mean a separate algorithm to handle that situation, it is to prefer.

3.2.3 Performance
All formulas and statements in this section come from [3].

From our point of view, the most interesting formulas are the ones showing the probability of the BC protocol avoiding blocking.

Let MTTF_c be the mean-time-to-failure of the coordinator and MTTF_b of the backup site. And let MTTR_c be the mean-time-to-repair of the coordinator; this is the service interruption of the coordinator. The probability P_c that the coordinator is down has the following formula:

$$P_c = \frac{MTTR_c}{MTTF_c + MTTR_c} \approx \frac{MTTR_c}{MTTF_c}$$

since MTTR_c << MTTF_c,

The probability P_b that the backup site fails is:

$$P_b = \frac{1}{MTTF_b}$$

Then we have the probability that both the backup site and the coordinator fails at the same time is:

$$P_b \times P_c = \frac{MTTR_c}{MTTF_b \times MTTF_c}$$

This formula tells us that the blocking in BC is reduced compared to 2PC, by the P_b factor. Even though the blocking is not removed completely, it is reduced significantly and could be acceptable in many cases. Reddy and Kitsuregawa, the authors of the BC protocol [3], have written another paper [4] where they use several backup sites and thereby reduce the blocking property even more, but that is not covered in this paper.

Simulations in [3] shows that BC performs closely to the basic 2PC protocol, which is good.

4. SUMMARY AND CONCLUSION
To avoid blocking in basic 2PC protocols some sort of non-blocking techniques is necessary. In this paper we have investigated two kinds of 2PC protocols with non-blocking features: N2PC and BC. They have different approaches to tackle this problem. N2PC does not send ACK messages and letting the participants take a bigger responsibility for the transaction. BC on the other hand, acts more like a 3PC protocol but the extra phase of messages is only between the coordinator and his backup site. In this way we reduce the amount of messages compared to 3PC.

We think that it is appropriate to regard N2PC as a non-blocking protocol since it handles all possible cases, except when all sites fail but one, as that exceptional case must be seen as a disaster and could be excluded.

The BC protocol could be compared to an ordinary 2PC protocol regarding transaction times. In table 1, this would mean 4(n-1) + 1 message exchanges, the extra message is to the backup site and with a lot of sites (big n) that would have little impact on the total transaction time.

In the future, a paper about comparing different methods of non-blocking techniques regarding 2PC protocols would be of great benefit for those who will use an atomic protocol for a distributed system. Especially in the performance field, now it is difficult to compare different techniques as they have used different methods to describe their work and results.

5. ACKNOWLEDGMENTS
We would like to thank Mats Björkman, Professor of Computer Communication at Mälardalen University, for consultation.

6. REFERENCES
[1] Two-phase commit protocol. Wikipedia. date 2009-09-30,


