Non-Atomic Commitment Problem: A comparative study between the 2PC and a new protocol based on the consensus paradigm

Tayeb LEMLOUMA [†] Nadjib BADACHE [*]

[†] *Opera Project, INRIA Rhône-Alpes, Zirst - 655 avenue de l'Europe, 38330 Montbonnot-St-Martin*
Phone: (+33) 4 76 61 52 81 Fax: (+33) 4 76 61 52 07

[*] *Institut d'informatique, U.S.T.H.B, B.P .32 El-Alia Bab Ezzouar, Alger 16111, ALGERIE.*
Tel / fax 213 2 24 76 07

Abstract: The atomic commitment problem is of primary importance in distributed systems, this problem become difficult to solve if some participants which are involved by the execution of the transaction commitment fail. Several protocols have been implemented to allow participants to terminate the commitment of transactions. In this paper we give a comparative study between the two phase commit protocol (which is blocking) and a new protocol that resolve the non-blocking atomic commitment problem using the consensus paradigm. The results of our comparison are based on a simulation of a set of sites that try to execute the commitment phase of a transaction in an asynchronous distributed system. In our implementation of the new protocol that we call the Consensus Commitment Protocol (in short CCP), we have used the solution of the consensus problem introduced by Chandra and Toueg and which is based on the concept of the unreliable failure detectors. The CCP used is based on the algorithm that uses the eventually strong failure detector noted $\Diamond S$. We presented also basic ideas to implement a failure detector of the $\Diamond S$ class, this later permit – when some properties hold – to resolve the consensus problem, we present also a new property of failure detection, which is simpler than the Eventual Weak Accuracy property. In addition to some basic concepts necessary to simulate an asynchronous distributed system. The distributed system simulated make easy to test distributed applications with any number of sites, and in every failure scenario we want.

Key-words: Atomic Commitment, Consensus Problem, Distributed Systems, Non-blocking Protocols, Transaction, Two Phase Commit Protocol, Unreliable Failure Detectors.

1 Introduction

The user data processing environment has been changed recently, by the progress in communication and database technologies. The actual data processing situation is characterized by a growing number of applications that require access to various local data sources located in heterogeneous hardware and software environments distributed among the nodes of a network. A multidatabase is composed of local data sources.

Systems that facilitate the logical integration of local data sources are called multidatabase systems. Applications accesses to data located in a local data source through *transaction*, this later can be viewed as an atomic group of invocations on data objects. In a multidatabase system, many sites can participate in the execution of the same transaction. At the end of the transaction, these participants, must execute a *commitment protocol* in order to guarantee a correct termination of the transaction.

The commitment protocol ensures the consistency of the distributed data, even if in presence of failure.

The *two Phase Commit* protocol, (in short 2PC) is one of the most protocols used in the atomic commitment. Unfortunately this protocol is blocking in some failure scenarios, more precisely : when the coordinator fails, and at least, one participant is waiting for the final decision. In this paper we are interested in the comparison of the 2PC and a new protocol that we implement, to resolve the non-blocking atomic commitment problem using the consensus paradigm. Consensus allows participants to reach a common decision (commit or abort the transaction), which depends on their initial inputs values (agree or not for the commitment) despite failures. We were based in our implementation of the new protocol on the consensus solution introduced by Chandra and Toueg and which is based on the concept of the unreliable failure detectors. Chandra and Toueg have shown that unreliable failure detectors can be used to solve Consensus in asynchronous systems with crash failures.

In order to compare the two protocols, we have simulated a distributed system, in which we have studied the behavior of each protocol in different cases, especially in presence of failures.

Our paper is composed of five sections. In Section 2, we introduce the atomic commitment problem and we show its importance in transactional systems. The classic solution (the 2PC approach), and the new one (the consensus approach), are given in Section 3and 4, respectively. In the last Section, we illustrate the obtained results of our comparison study between the 2PC and the CCP, that we implement in a simulated distributed system. We start by describing our simulation and its objective. Then, we give our proposition to implement a failure detection module. We discuss the practical possibility of solving atomic commitment problem in asynchronous distributed system; and we also propose a new failure detector property which is easier to guarantee than the Eventual Weak Accuracy. In the rest of the section, we compare the behavior of the two studied protocols, in terms of commitment time and fault tolerance degree. Our paper can be seen as an application of the theoretic concept of failure detectors, and an evaluation of the efficiency of a new protocol based on the consensus paradigm, compared with the protocol which is generally used in distributed transactions commitment.

2 The Atomic Commitment Problem

The multidatabase system (MDBS) consists of a set of local database management systems(DBMS) that manage different local data sources [1]. Access to data located in a local data source is accomplished through *transaction*. A transaction is a portion of a user program written in a high level programming language, it is an organized collection of such operations that from the point of view of the application accomplishes a logically indivisible unit of work. A fundamental assumption of the transaction concept is that correctness is defined by the application, i.e., when a transaction is executed alone it preserves database consistency. Database systems rely on the transaction designer (who understands the application constraints) to make sure that each individual transaction does not violate the database consistency.

Generally, an atomic action *A* is an operation that changes a set of object from an initial state to a final state. All the objects concerned by the action are inaccessible during the execution of *A*. These objects return to their initial state, if a failure occurs before the termination of the atomic action.[9]. In a distributed system, the atomic commitment problem consists of ensuring that all correct participants of a transaction take the same decision, namely commit or abort the transaction. If the decision is Commit then all participants make their updates permanent; if the decision is Abort then no change is operated on the data (the transaction has no effect). The Commit/Abort value of the outcome of the NBAC protocol depends on the votes of participants and on failure.

3 Solution based on the 2PC

In a distributed algorithm designed to achieve a global objective, each processing site has to carry out its computations and actions based on the information available to it, in order to achieve the global objective. Therefore, the algorithm has to be designed to co-ordinate the processes in such a way that the proper consistent information is available to each processing site at the proper time.

The Two Phase Commit protocol(in short 2PC), is the simplest and the best known protocol, it that has as object to ensure the atomic commitment of a distributed transactions, it is based on a centralized control mechanism contains a single process , called the *co-ordinator*, which co-ordinates the actions of the others.

The co-ordinator sends a transaction request ton the others and waits for their replies in the first phase. After receiving all applies, the co-ordinator sends a final decision to the others in the second phase.

Unfortunately, the two phase commit protocol is blocking. For example, if the coordinator fails while Data Manager processes are waiting for a decision message, then none of these processes can terminate the transaction. A process must wait for the coordinator to recover before deciding on an outcome.

4 Solution based on the Consensus

To solve the atomic commitment problem, we were based on the consensus paradigm, the consensus can be used as a building block in many other problems that arise in practice, such as its application in distributed mobile environment introduced in [10]. Consensus allows processes to reach a common decision, which depends on their initial inputs, despite failures.

As any decentralized solution, the consensus solution introduced by Chandra and Toueg, does not use a single coordinator. All participants are considered to be identical. In every round, all participants execute the same program, send and receive a set of messages. Since there is no central controller, the information required for a site to accomplish the global objective cannot be obtained by exchanging messages with one site only.

Chandra and Toueg have introduced the concept of unreliable failure detectors and shown how they can be used to solve Consensus in asynchronous systems with crash failures. They have proved that Consensus can be solved even with unreliable failure detectors that make an infinite number of mistakes. In few words, an unreliable failure detector is a module which outputs the set of processes that it currently suspects to have crashed. Such detector is specified in terms of two abstract properties that it must satisfy: *Completeness* and *Accuracy*. Completeness requires that the failure detector eventually suspects every process that actually crashes, while accuracy restricts the mistakes that a failure detector can make.

In our implementation of the *Consensus Commitment Protocol*, that we call: CCP, we were based on the algorithm which uses the *Eventual strong failure detector ◊S*[3].

This algorithm is based on two basic ideas:

1-Te rotating coordinator paradigm[13, 4, 5, 11, 2], which was already used in some protocols, such as the *three phase commit protocol*.
2-The concept of unreliable failure detectors.

In the next section, we evaluate the efficiency of the 2PC and the CCP. We describe our model of asynchronous distributed system, and give some basic ideas, necessary to simulate such model, and to implement our protocol in such environment. We also expose some experimental results from our simulated system, in order to see better how each protocol behaves in some cases, especially when some failure occurs.

5 Experimentation and results

A distributed system is a collection of sequential processes communicating solely by exchanging messages. In such system the behavior of each process is completely determined by a local algorithm which also defines the process reaction to incoming messages. The process may then change its state and eventually sends messages to other processes.[10]

The implementation of distributed algorithms in a centralized environment (for example in an execution environment with a single processor machine), requires simulation tools, which ensure the distribution, as in a real distributed system.

We consider an asynchronous distributed system, in which processes are completely connected. Communication between processes is ensured through reliable links. A process may fail by crashing and a correct process is a process that does not crash during the course of an infinite run.

5.1 Simulation

Our simulation has as object, to preserve all the characteristics of a real asynchronous distributed system, i.e.:

1- The absence of a common shared memory.
2- The absence of a global clock.
3- The existence of a communication system, which must ensure communication between processes.
4- The absence of bounds on message transmission delays and relative process speeds.

5.1.1 Sites description

Each site of our system is represented by a process which is executed simultaneously with the other processes. This simultaneous execution, simulates the fact that sites behave independently in the reality. Indeed, none shared information or variable exists between every pair of process, the unique way of communication, is by exchanging messages, exactly as in a real distributed system. A particular process which we call *the communication process*, ensures the task of communication between every pair of processes.

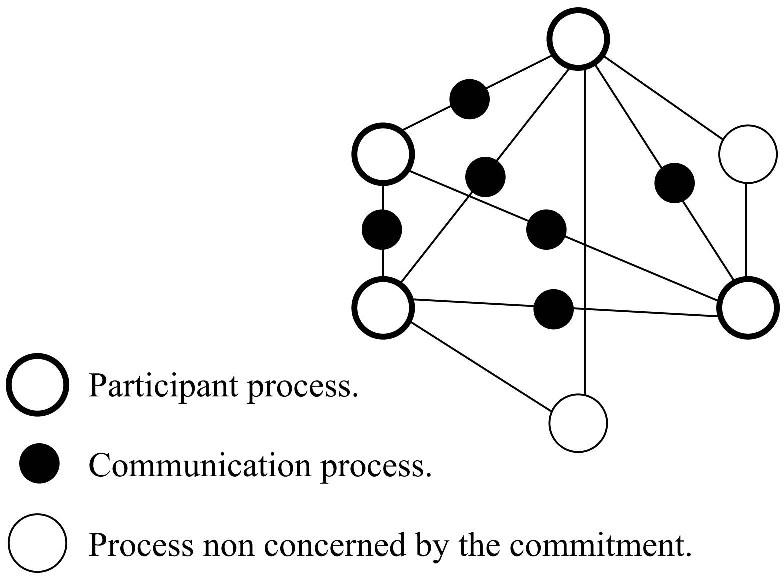

Figure 1: The simulated distributed system.

Our simulation permit to test the efficiency of protocols with any number of process, and to study easily their behaviors in different situations and with different parameters of the distributed system.

5.1.2 Communication between sites

In our simulation, a particular process, called *the communication process*, plays the role of a communication link, which exists between two given processes. The communication process has a unique task which is ensuring message transfer. The creation of such process is necessary, because we must simulate the fact that there is no bound on message transfer delays and this independently to the execution of the other process.

The algorithm of the communication process must ensures assumptions made on the network, i.e.:

1-The reliability of the communication: The communication is always possible between each pair of operational processes.
2-The absence of loss and deterioration of messages: primitives of communication used are reliable.
3- The order of message transmission is the same than the order of reception.

The communication process executes the following algorithm:

```
□The communication process algorithm

Cobegin

 || Task 1 :repeat forever
  when  receive message from p
  wait (delay1) ;
  send (message) to q ;

  || Task 2 :repeat forever
  when  receive message from q
  wait (delay2) ;
  send (message) to p ;

Coend;
```

The communication process must modify *delay1* and *delay2* to simulate the variability of message transfer delays.

5.2 Failure detectors

The CCP uses a particular module which plays the role of a failure detector. Every participant, consults its failure detector module, this later outputs the set of process that it currently suspects to have crashed.

Since our protocol is based on the $\Diamond S$ detection class, the detector module must satisfy the *Strong Completeness* and *Eventual Weak Accuracy* properties [3], in its detection.

To ensure the Strong Completeness property, we must ensure that the failure detector module of every correct participant, suspects permanently every participant that crashes. To implement this, we were based on the *time-out technique*: If a participant *p* times-out (expires its time-out) on some participant *q*, it adds *q* to its set of suspects, and it broadcasts a message to *all participants* with this information. If a participant receives this broadcast, then it adds *q* to its set of suspects. If *q* has not crashed, it broadcasts a refutation. When a participant receives *q*'s refutation, it removes *q* from its set of suspects. This technique guarantees the Strong Completeness property as it was defined by Chandra and Toueg. Note that the failure detector can suspect erroneously a correct participant (for example, if this later is very slow), this is why we qualify such detector by "unreliable".

The following algorithm, abstracts the time-out mechanism. This algorithm is executed by a process *p* which wait for a message from an other process *q*:

```
□ Time-out mechanism

Procedure wait_message(q) ;
{wait for a message from q}
 Begin

   time-out := time-out_estimated; {Activation of the time-out value}
  {After 'time-out_estimated' unity of time, the system initializes the time-out value}
    reception := false ;
    while (time-out not expired) and ( reception = false) do
    if (the message is received from q) then reception := true ;
   if (reception = false) then suspected = suspected ∪{q} ;

 End ;
```

The second property of the $\Diamond S$ class, which is the Eventual Weak Accuracy, is more difficult to ensure than the first property. To satisfy this property, the detector module must ensure that *"there is a correct process and a time after which that process is not suspected to have crashed "*. This property is difficult to satisfy, because at any given time *t*, participants cannot determine whether any specific participant *p* is operational, or whether some operational participants will never be suspected after time *t*.

In the commitment phase we need that the distributed system verifies some properties merely during a certain period of time, these properties allow the CCP to terminate the commitment of the distributed transaction.

To be more exact, the behavior of the distributed system – composed of the set of participants in the commitment – must be the same as the behavior of the models of distributed systems in which consensus is solvable.

In practice this situation can holds for three reasons :

1- In the commitment phase, we are interested by the distributed system which is only composed by the set of participants. So all the rest of the sites is not concerned.
2- The behavior of asynchronous distributed systems is not deterministic i.e.: it can changes on the time in terms of bounds on relative process speeds and on message transmission times.

3- The period of time wanted is limited.

More details on the models of distributed systems are in [6]. [6] gives four models in which consensus problem is solvable, for example, let us consider a system in which participant speeds and message transfer times become bounded after some time *t* and during some time *d*, note that we don't assume that bounds are known
and the same thing for *t*. Our implementation of the CCP ensures a correct and consistency termination of the transaction commitment in such system.

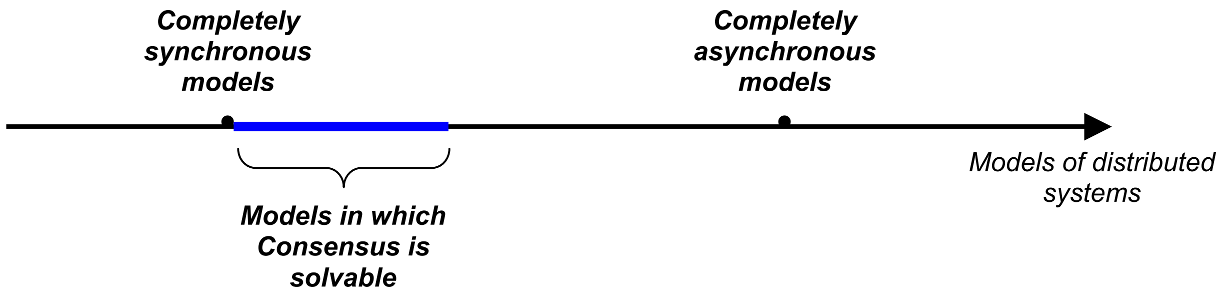
Completely synchronous models
Completely asynchronous models
Models of distributed systems
Models in which Consensus is solvable

Figure2 : The Consensus problem and the models of distributed systems

We introduce in which follows, a new property, which can replace the Eventual Weak Accuracy in the transaction atomic commitment. When this property is satisfied, the CCP can terminate easily the commitment.
If we follow Chandra and Toueg's notation, we can write our property, that we call *the Commitment Eventual Weak Accuracy property*, as follows:

$\forall F, \forall H \in D(F), \exists t \in T, \exists p \in correct(F), \forall t' \; t \leq t' \leq t+t_0, \forall q \in \text{correct}(F)$:
$p \notin H(q, t')$. Thought that $d = (t_0 - t)$ is the time necessary to terminate the transaction commitment.

This property is easier to satisfy than the *Eventual Weak Accuracy.* However the problem of its implementation persists always. We can propose to give a high value to time-out_estimated in order to avoid erroneous suspicions (this is what we have used in our implementation of the failure module). This proposition is not perfectly efficient, especially in purely asynchronous systems which are characterized by the absence of bound on communication delays.

5.3 Evaluation of the two protocols

The practical advantage of the Atomic Commitment Protocol based on the Consensus problem is the absence of blocking. Of course, even if a failure occurs, all the non-failed participants can reach a correct decision. On the other hand the two phases commitment can block the system if only one site fail.

Our protocol solves the non-atomic commitment problem in a distributed system augmented by an eventually strong failure detector.

We illustrate in the following, the obtained results after a wide number of tests on the two protocols behaviors, and that in our simulated distributed system. Tests were done in a single machine processor, running under the Unix system.

5.3.1 Commitment time

In this section, we will compare the 2PC and the CCP, in terms of execution time, i.e. in terms of the time necessary to terminate the atomic commitment of a given transaction. Don't forget, that in our distributed system model, there is no common clock in the network. Thus none participant can participate to the measurement of the transaction commitment time, moreover this measurement requires the knowledge of the global state of the network and this at any given time *t*. Indeed in our simulation, we use a particular process, called the *supervisory process*, which plays the role of a global clock. The supervisory process can be seen as a fictional device that *participants can not access to it*. The supervisory process calculates the time which separates the moment of the beginning of the commitment protocol, and the moment of the effective commitment of the transaction

The values given in which follows, are average values of several experiments done on a single processor machine. Note that for a same commitment scenario and with the same parameters of the network (message transmission times, time-out, number of sites... etc.), we can obtain different values of commitment time, this is mainly due to the execution environment (only one processor, the scheduling strategy which assigns the CPU to the processes in a non deterministic way,... etc.), moreover the non determinism of the distributed algorithms used. However the values measured give a clear idea of the ACPs commitment way and of the commitment duration in a real network.

The following table, gives - in the absence of failures – average values of the atomic commitment time, using the 2PC and the CCP, in different cases.

Number of sites	Votes	**Commitment time** (unity of time)	
		2PC	**CCP**
2	(1)	1	2
2	(2)	1	2
2	(3)	1	2
4	(1)	2	4
4	(2)	2	4
4	(3)	1	4
7	(1)	3	6
7	(2)	3	5
7	(3)	2	5
10	(1)	5	8
10	(2)	5	8
10	(3)	4	7

Votes **:** (1): All the sites vote "YES".
(2): Some sites vote "YES" and some other vote "NO".
(3): All the sites vote "NO".

Figure 3: A comparative table of the commitment time (without failures).

Comment :

In the CCP protocol the concept of the fixed coordinator does not exist, all the sites take part in the same way in the atomic commitment. For this reason, all the sites must communicate to terminate the commitment and to reach a consistent decision. On the other hand, in the 2PC there is only one site (the coordinator) which is responsible to calculate the global decision of the commitment, therefore the collection and the comparison of votes, and the diffusion of the final decision is done on a same site, this is mainly which justify that in the absence of failures, the CCP make more time to terminate the atomic commitment of a transaction than the 2PC protocol, as it is shown in the preceding table.

In short, the centralized control make the 2PC more interesting compared to the CCP, in the case where no participant fail, which is - generally - rare in distributed systems.

In the following table, we compare the average time of the atomic commitment for the two studied protocols, and that in the presence of failures.

Number of sites	**Votes**	**Number of failed sites**	**Commitment time** (unity of time)	
			2PC	**CCP**
2	(1)	1	$+\infty$	(*)
2	(2)	1	$+\infty$	(*)
2	(3)	1	1	(*)
4	(1)	1	$+\infty$	8
4	(2)	1	$+\infty$	8
4	(3)	1	1	8
7	(1)	1	$+\infty$	16
7	(2)	1	$+\infty$	16
7	(3)	1	2	10
7	(1)	3	$+\infty$	19
7	(2)	3	$+\infty$	19
7	(3)	3	1	12
10	(1)	2	$+\infty$	17
10	(2)	2	$+\infty$	17
10	(3)	2	4	15
10	(1)	4	$+\infty$	28
10	(2)	4	$+\infty$	28
10	(3)	4	3	27

Votes **:** (1): All the sites vote "YES".
(2): Some sites vote "YES" and some other vote "NO".
(3): All the sites vote "NO".

Commitment time : (*) It can't be calculated, because the number of failed sites is not supported.

Figure 4: A comparative table of the commitment time (with failures).

Comment :

We have shown that in the presence of failures, the 2PC is blocking, this means that participants can neither commit the transaction, nor abort it. On the other hand, the protocol based on the consensus avoids blocking even if there are some failed sites.

The above table, shows clearly the advantage of the CCP, compared to the 2PC. Indeed the CCP presents a maximum of fault tolerance contrary to the 2PC which can block if only one site fails. Note that if the time of commitment is equal to $+\infty$, then that means that the protocol is blocking in the corresponding case. *Example*: the 2PC is blocking if all the participants vote "YES" and one site (the coordinator) crashes. (see the above table, *line 1*).

As one can notice, *line* 1 to 3 in the above table, the CCP can not tolerate the given number of failed sites. That is logical, because the ACP based on the $\Diamond S$ failure detectors class, requires a majority of operational participants, so if n (2 in our case) is the number of participants, we must have $\lceil (n+1)/2 \rceil$ correct participants (2 participants in our case, therefore no site must be failed). On the other hand, the 2PC does not pose any preliminary restriction on the correct participants number. *Example*: Although there is not a majority of correct participants, the 2PC terminated the commitment without blocking in the case of the third line, (in the above table).

We can also notice that in the presence of failures, the CCP atomic commitment time is higher, then the time in the normal case, where all the sites are correct (see the two above tables, FIG xxx and FIG xxx). This is justified, by the fact that our implementation of the failure detector module, is based on *time-out detection mechanism*, which is the unique possible mechanism to implement such modules in asynchronous distributed systems. Thus if a coordinator c of a round r crashes, then it will not be detected only after the expiration of *time-out-estimated* unity of time, see the third phase of the consensus solution based on the $\Diamond S$ failure detectors class [3]).

5.3.2 Fault tolerance

More a distributed algorithm tolerate failures, more it is interesting. This criterion is related to the level of symmetry[12]: if a process plays a particular role, the algorithm is sensitive to failures, on the other hand, if all the processes play the same role, the algorithm can operate even if some failures occur. The algorithm of Chandra and Toueg which resolve the consensus problem is based on this basic idea. We can classify this algorithm in the calculation algorithms of functions class [8]. Indeed, the algorithm can be seen as a function which calculates in a distributed way a result which is here the commitment or the abortion of the transaction.

The atomic commitment protocol which is based on the consensus, requires only a majority of correct participants. If this condition is satisfied, the protocol is non blocking: it terminates the commitment without any risk of blocking nor of an inconsistency decision.

To see better the fault tolerance degree of each studied protocol, let us use probabilistic tools : Let us assume that the event "to crash", has the same probability for each process that participate in the transaction commitment, let p is this probability. We also assume that participants are independents from the point of view of this event.

If n is the number of participant processes in the transaction commitment, and $\alpha = \lfloor (n+1)/2 \rfloor$, the following table gives blocking probabilities for each studied protocol: the 2PC and the CCP based on the $\Diamond S$ failure detector class. Probabilities were calculated using the independent events rule : "*if A and B are two independent events so, $P(A \cap B)=P(A).P(B)$.*". p : is the probability that one participants crashes, p^{α} : is the probability that there is not a majority of correct participants, thought that α indicates the majority of the transaction participants.

	2PC	CCP
Blocking Probability	p	p^{α}

Figure 5: Probabilities of blocking.

We can notice, from this table, that the probability of blocking of the commitment protocol based on the consensus, is less than it of the two phase commit protocol (because $0 \leq p \leq 1$and $\alpha > 1$), this fact, disadvantages the use of the 2PC.

In the absence of failures, the consensus permit to finish the transaction commitment during the first round of the execution of the *propose* primitive (see the solution based on the $\Diamond S$ class [3]). In that case, the behavior of the CCP is similar to the 2PC's behavior.

During the first round:

- Participants send their estimate values (or their votes) to the current coordinator *c*.
- After the reception of the votes, the coordinator *c* of the first round (round $r = 1$), sends its estimate, and according to participants replies, the coordinator reliably broadcasts the final decision (or the estimate value) to all participants. Note that in the protocol based on the consensus, the coordinator does not need to wait to all the estimate values, a majority is enough to broadcast the decision.

It is only in the case when failures appear, that the CCP behaves differently than the 2PC. In this case, the CCP avoids the blocking by the run of a number of rounds, in order to take a final decision (termination of the execution is ensured).

The number of the exchanged messages depends on failures suspicions. If there are neither failures, nor failures suspicions, the following table gives the complexity of the algorithm which solves the problem of the consensus using the $\Diamond S$ class.

	Rounds Number	**Messages number**
CCP	*3*	*3(n-1)*

Figure 6: The complexity of the consensus solution(using $\Diamond\mathbf{S}$*).*

Note that we must add the number of the exchanged messages, required by an execution of the reliable broadcast primitive [7], to the total number of the exchanged messages.

6 Conclusion

Several atomic commitment protocols have been proposed in the literature, unfortunately most of them are blocking in case some participants crash.

In this paper we gave a comparative study between the two phase commit protocol and a new commitment protocol based on the consensus solution introduced by Chandra and Toueg, this solution uses the $\Diamond S$ failure detector class. In our protocol, we tried to implement the theoretic concept of the failure detector in an asynchronous distributed environment. Experimental results – which are based on a simulated distributed system – have shown that our protocol behaves better than the 2PC especially in the presence of failures.

The atomic commitment protocol based on the consensus paradigm, solves the blocking problem in the asynchronous distributed systems. This protocol:

- Does not pose many assumptions on the network.
- Terminates correctly the commitment phase of transactions, in *asynchronous* distributed systems.
- Tolerates failures and requires only a majority of operational sites, which is, generally, easy to ensure in practice.

These advantages allowed by this new protocol, encourage us to defend two ideas:

- The practical use of this new protocol in the network, for the transactions atomic commitment.
- The exploitation of the consensus paradigm and its solution introduced by Chandra and Toueg, to solve, practically, other agreement problems in asynchronous distributed systems.

Acknowledgment

We are deeply grateful to Tushar Deepak Chandra for his help on the explanation of the consensus problem and the algorithms that use the *S* and *◊S* failure detector class.

References

[1] Bernstein Philip .A. , Hadzilacos Vassos. , Godman Nathan.
Concurency control and recovery in database systems.
Addision-Wesley, Reading, Massachusetts. 1987, 370 page.

[2] Chandra T.D. and Sam Toueg.
Time and message efficient reliable brodcast.
In procedings of the fourth international workshop on distributed algorithms, pages 289-300. Springer – Verlag, September, 1990.

[3] Chandra T.D., and Toueg S.
Unreliable failure detectors for reliable distributed systems.
Journal of the ACM, 43(2), pp: 255-267, (March 1996).

[4] Chang J., and N. Maxemchuk.
Reliable broadcast protocols.
ACM transactions on computer systems, 2(3):251-273, August 1984.

[5] Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer.
Consensus in the presence of partial synchrony.
Journal of the ACM, 35(2): 288-323, April 1988.

[6] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer.
On the minimal synchronism needed for distributed consensus.
Journal of the ACM, 34(1):77-97, January 1987.

[7] Hadzilacos V., Toueg S.
Reliable Broadcast and Related Problems.
Distributed Systems (*Second Edition*), ACM Press (S. Mullender Ed.), New-York, 1993, p. 97-145.

[8] Helary J.M. Raynal M.
Vers une problématique de l'algorithmique réparti
Publication interne #471, Mai 1989. IRISA.

[9] Krakowiak Sacha.
Principe des systèmes d'exploitation des ordinateurs.
Bordas, Paris, 1987.

[10] Nadjib Badache, Michel Hurfun, Raimundo Macedo.
Solving the consensus problem in a mobile environment.
Technical Report #1146, IRISA, Rennes, November 1997.

[11] Pior Berman, Juan A. Garay, and Kenneth J. Perry.
Towards to optimal distributed consensus.
In procedings of the thirtieth symposium on foundations of computer science, page 410-415. IEEE computer society press, October 1989.

[12] Raynal M.
Algorithmes distribués.
Eyrolles 1985.

[13] Rudiger Reischuk.
A new solution for the Byzantin general's problem.
Technical report RJ 3673, IBM research laboratory, November 1982.