Atomicity in Mobile Networks

Joos-Hendrik Böse[1], Stefan Böttcher[2], Sebastian Obermeier[2], Heinz Schweppe[1], Thorsten Steenweg[2]

[1] Freie Universität Berlin, Institute of Computer Science, Takustr. 9, 14195 Berlin, Germany
boese@mi.fu-berlin.de, schweppe@mi.fu-berlin.de
[2] University of Paderborn, Computer Science, Fürstenallee 11, 33102 Paderborn, Germany
stb@uni-paderborn.de, so@uni-paderborn.de, steenweg@uni-paderborn.de

Abstract: Atomicity, one of the essential transaction properties, is guaranteed in fixed-wired network by protocols, like the 2-phase commit protocol, which assumes that faults like node and link failures rarely occur. However, node and link errors which are considered as an exception in fixed-wired networks can be assumed to be the default case in mobile networks. Therefore, depending on the application, mobile networks require different protocols for guaranteeing strict atomicity. Within this paper, we look at different application scenarios and their requirements w.r.t. atomicity. We distinguish classes of scenarios in which atomicity can be guaranteed from classes of scenarios in which atomicity cannot be guaranteed. Furthermore we will show what kind of transactional guarantees can be realized in the different scenarios.

1. Introduction

1.1. Motivation

Providing transaction processing with guaranteed transactional behavior in volatile environments is a difficult task. Constant change in available connections and frequent link failures leave little to build guarantees on.
Atomicity has been studied in different areas of computer science and is fundamental to simplify reasoning about the correctness of distributed applications and dealing with concurrency and fault tolerance in complex systems, like distributed databases, peer-to-peer systems and service oriented architectures. The notion of atomicity in this work is understood as a means to maintain consistent states in distributed systems, because any execution schedule of atomic operations will result in a correct state of the overall system. We will only consider strict atomicity; concepts like relaxed atomicity or semantic atomicity are beyond the scope of this work.
In distributed systems communication between participants of atomic operations is needed to agree on the new correct state of the overall system. If communication between the participants is assumed to be asynchronous and unreliable, certain problems like the distributed attack problem [Gr78] are unsolvable. Although protocols like the 2-phase commit protocol show a blocking behavior [BHG87], they are found to be practically applicable, because in contrast to mobile environments,

these cases of blocking are very rare in fixed networks and can be recovered using extensive recovery strategies. So they do not significantly affect the operability of the system.
However, in mobile networks we assume link failures to occur frequently. Hence in such environments, protocols like the 2-phase commit protocol are not applicable as blocking must be expected as the normal behavior of the system and not as the exceptional case. Nevertheless, applications deployed in mobile networks form complex distributed systems, which have strong demands for atomicity in terms of agreements on new states. Examples for such atomicity requirements are money atomicity and goods atomicity as introduced by [Ty96].
In this paper we will describe what atomicity guarantees can be given in mobile networks for different kinds of applications.
Settings like the coordinated attack problem are well understood and proofed to be unsolvable in case of link failures and asynchronous communication as shown in [Gr78]. Hence we will weaken the requirements of applications w.r.t. atomicity and show what kind of transactional atomicity guarantees can be realized.
We will set up a specific system-model and show which atomic guarantees can be given within identified classes of applications and coordination problems. We will show that certain classes of applications do not depend on requirements of the distributed attack problem with link failures.
We will identify characteristics of such problems and show how atomicity can be achieved.

The remainder of the paper is organized as follows: Section 1.2 reviews the research done on the problem of distributed consensus with link failures and summarizes the current state of the art. Section 2 will describe the basic assumptions of our system-model. In section 3 two different classes of application scenarios are described, each representing a class with common requirements. Section 4 classifies problems as well as the atomicity guarantees that can be provided. Finally section 5 concludes and summarizes this work.

1.2. Related Work

The impossibility of the deterministic version of the coordinated attack problem was proved in [Gr78]. Approaches to solving this problem by making some probabilistic assumptions about the loss of messages while keeping the processes deterministic have been tried. Another approach, described in [VL92], allows the processes to use randomization, and considers the possibility of violating the agreement and/or validity condition. Both approaches guarantee to reach correct agreements.
Activities that directly approach the basic problem of unreliable links are approaches using cross-layered designs, as the layered approach to computer networks introduces an abstraction level that loses useful information about the connection context. But this information can be used to calculate parameters, e.g. the probability of message loss. For example communication with nodes that move in different directions is more likely to fail than a link between nodes moving in the same direction. The same idea can be applied to information about energy resources as done in [FG04].

How to design distributed data processing systems in fixed networks is well researched in [Ly94]. Most conventional distributed databases deployed for fixed-wired networks use the 2-phase commit protocol ([Gr78]) or variations like the 3-phase commit protocol ([Sk81]) to achieve an atomic commit of distributed transactions. Relaxing strict atomicity to semantic atomicity has led to optimistic 2-phase commit protocols like [LKS91]. Protocols like [HRG00] propose a commit protocol for transactions with timing constraints, so-called real-time transactions for distributed systems. In contrast to [Li02] these protocols were designed for fixed-wired systems. A protocol for real-time transactions in mobile environments is proposed in [Li02]. However, this protocol offers only relaxed atomicity and not strict atomicity.

Most work about transaction processing in mobile environments like [DHB97, WC99, Ch93] assumes a system model where transactions are processed between mobile clients and a database server residing in a fixed network, that can be reached via a base station that is connected to the network. Transaction processing between mobile nodes is not the scope of these activities.

A timeout based approach to atomic commit decisions is proposed in [Ku02], where the coordinator decides for global commit if he does not receive a failure message from a node in the commit set within a predefined timeout period. Mobile nodes can tell the coordinator to extend this period if they need an extension. The main problem here is to calculate an appropriate value for the time-out period as it depends on a number of system variables which could be difficult to quantify.

2. Assumptions of our system-model

In our system-model we assume that mobile nodes communicate with each other directly using some kind of wireless technology. Because of node movement, direct communication between mobile nodes is very unreliable, i.e. every node and every link may fail at and for an unforeseeable time. Mobile nodes can also communicate with mobile support stations that are connected to a fixed-wired network. We assume that failures of the fixed-wired network occur rarely and especially that the commit decision information that is stored in the fixed-wired network will not get lost. However, due to movement of the mobile client, communication between the fixed-wired network and the mobile clients is also unreliable.

Furthermore, we assume that when a device reconnects to the system after a link failure, it can check the status of the transaction it was processing when the failure occurred, i.e. we assume that commit status information of transactions is stored at a safe place (e.g. within the fixed-wired network).

3. Application Requirements and Scenarios

This section describes general requirements to atomicity and two classes of application scenarios which differ in their atomicity requirements. We use these scenarios to identify common characteristics and to classify the addressed problems.

3.1. General Requirements to Atomicity in Mobile Networks

The requirements of atomic commit protocols can be grouped into general requirements which are common to all our mobile application scenarios and specific requirements which additionally apply only to some of the scenarios. The general requirements are mostly similar to atomic commit protocols in fixed-wired networks. These requirements include:

1. The atomic commit protocol must direct all participating processes to the same decision.
2. If a participating process disappears irrevocable (e.g. a mobile device gets lost or damaged) before its vote is received by a coordinator, its vote has to be treated as for abort.
3. No process can withdraw its decision. This includes dying processes, whose votes have already been received by a coordinator.
4. The global decision must be to commit, if all involved processes vote for commit.
5. If all failures are recovered and no new failures occur for a sufficiently long time, every process reaches a final decision.
6. If no failures occur, all messages are delivered without delay, and all participants vote for commit, the protocol must decide for commit.
   Note that this last requirement is more relaxed than the usual requirements for atomic commit protocols because we do not require a common decision to be commit if messages do not arrive in time.

Another group of general requirements limits the effects of link failures to the directly involved nodes. Hence we define further requirements:

7. No party shall reside in a state of uncertainty for an arbitrary long time due to link failures of another node. More precisely, when a link from a mobile partner to a safe coordinator fails, at most this mobile partner should be blocked. However, other participants, that have voted for a commit, should be able to continue their work before the failing link is recovered.
8. Similarly, when a mobile node participating in a transaction fails, running participants with a working link to a coordinator should not be blocked.

3.2. Class 1: Recursive Atomic Decision Scenarios

Within the first class of atomic commit scenarios, the individual participant's decision depends on whether or not the decision is guaranteed to be communicated in time to the other parties. We call these atomic commit decision problems *recursive atomic commit decisions*.

A well known example for such a recursive atomic commit decision is the coordinated attack problem outlined in [Gr78]. Here, two generals must agree on a time for a common attack using an unreliable communication channel. Within this scenario, a commit decision is not possible under the assumption of message loss. The reason therefore is that the decision of one general is recursively dependent on the decision of the other general. The decision is recursively dependent in the sense that one party can only promise to attack at a certain time only if that party knows that the other party is going to attack at the same time, which implies that the first party knows that the second party knows that the first party is going to attack at this time. The main problem here is that because of the unreliable communication no party can be sure that messages containing a commitment to a certain time are received by the other party before that time. Thus the origin of this recursive relationship lies in the demand to agree on a certain time to attack.
An example from the real world involves two cars exchanging information (e.g. about traffic conditions) while passing each other. A car that provides useful information for the other driver wants some kind of electronic cash in return. Communication is limited and unreliable in the sense that information can only be exchanged within a short time frame. Furthermore, we assume that there is no global coordinator available. A car is only willing to pay if it will get the information and vice versa. A car will not pay if it's not sure that it will receive the information paid for (and vice versa information is not uncovered if the car is not sure that it gets paid). There is no guarantee that a message reaches the other party within the time that the distance between the two cars allows for communication. The commit decision is recursive in the sense, that the decision to pay or to uncover a piece of information depends on the knowledge that payment or information reaches the other party.
A similar situation occurs between mobile nodes that want to exchange electronic goods between each other, if we assume that nodes do not trust each other and therefore, no node wants to send its items first since it must assume to get nothing in return. Note, that in the cars' scenario the problem of a recursive decision occurs even if the parties trust in the "correct behavior" of the other party, but cannot trust in the channels used to transmit their messages, i.e. there is no guarantee that their messages are delivered in time.
Within recursive commit decisions (i.e. class 1 scenarios), the commit decision result depends on whether or not this decision is guaranteed to reach the other parties within a given time. In other words, one requirement is to come to a decision before a deadline is reached *and* the result of the decision depends on whether or not the deadline can be met.

3.3 Class 2: Independently Completing Transactions

The characteristic of this class is that after agreeing on a commit decision, the transaction can be executed on each client, independently of the other clients' execution status. The transaction can be executed immediately on each client after a commit decision has been received.

To emphasize the difference between this class of scenarios and the recursive class 1, we create a slightly modified coordinated attack problem. In this scenario, the two generals do not need to agree on a certain attack time, but on a capitulation decision: e.g. raise a white flag for capitulation or a black flag for continuing the defense. It is not necessary that these flags are raised at the same time, but all parties must agree on the same decision. This means the siege is over only if all parties raise the white flag.
Another scenario which applies to this group is the trading of electronic goods in a trusted environment. Within this scenario, information is sold for E-cash. Assuming that all partners within this scenario trust each other, the decision to send an item (E-cash or information) first has no effect on the outcome, as every node trusts in receiving items in exchange.
Another scenario we have in mind is the scenario where a number N of mobile customers, that meet in a store, pool together in a corporate buying transaction to attain a volume-N discount. It must be guaranteed that each individual customer voting for "buy" deposits his share of the payment at a coordinator. In addition, he should not be able to withdraw his buying decision. If all the N customers come to the decision "buy", the discount buying transaction can be completed – otherwise the buying transaction is aborted and the mobile customers get their money back. The decision is not recursive if we assume that all customers are willing to wait until they are informed about the buying decision.
When a node disappears forever (i.e. a mobile device is irreparably damaged) after his vote was received by the coordinator, we assume that the transaction state reached on this mobile device has no influence on the transactions states of the other involved nodes. However, the other mobile devices participating in the transaction have to follow the commit decision, if the commit status has been decided.

4. Atomicity in Mobile Environments

Assuming the requirements of the system-model described above are fulfilled, this section will show what kind of atomic guarantees can be realized in such an environment and what kind of guarantees cannot be achieved.
For all class 1 scenarios, atomic commit decisions are limited by guaranteed message delivery times in the following sense. If a maximum time interval within which message delivery can be guaranteed does not exist, a safe atomic decision for commit is not possible. In other words, the only safe decision is to abort – however, this prevents transactions from being processed and therefore is not desirable. Timeout based protocols can also not guarantee a correct strict atomic commit decision.
For all class 2 scenarios, an atomic commit protocol that fulfills the requirements exists, i.e. when a node disappears forever (i.e. a mobile device is irreparably damaged), we assume that for the correctness of the overall application it does not matter which transaction state is reached on this mobile device. However, the other mobile devices participating in the transaction have to follow the commit decision if the commit status has been decided. If the node unexpectedly recovers it must follow the global decision. The difference to scenarios of class 1 is that none of the involved nodes has to commit itself to finish a local transaction at a certain time. The promise a

participant makes here is to commit if a coordinator tells the participant that all the others voted for commit, regardless of any time constraints. Class 1 scenarios, in contrast, cannot make this promise due to the recursive character.

While in principle the 2-phase commit protocol of distributed databases could be used for class 2 scenarios, this protocol has the following major disadvantage. Every local transaction locks resources as long as it has voted for commit and the commit decision has not been confirmed or revoked by the coordinator. Furthermore, the coordinator's decision depends on the votes of all participants. However, in a mobile network the vote messages are likely to be delayed due to link failures. In such a case, all other participants would have to wait for the vote of a single participant which is unavailable. In contrast to this, it is desirable to finish a transaction as fast as possible, respectively to reduce the probability of blocking, equivalently to striving for independent recovery under as many circumstances as possible. One possibility as also described in [UG01] is to associate the voting-phase with a time-out as follows. A commit coordinator chooses an appropriate time-out period. Whenever one of the participants or a communication link fails such that the commit coordinator does not get all the votes during that time, the coordinator treats the node from which it has not received a vote as if it had send an abort message. Otherwise, the coordinator has all votes and proceeds according to the protocol. Unless we use a cross-layer approach that does not hide status information of mobile devices from an application, it is impossible to determine if a mobile client will recover or has disappeared forever. Therefore this time-out based mechanism is one of the few promising approaches we see to reduce blocking. Hence it is important to find an optimal time-out period. Dynamically calculating this value based on the connection context of mobile clients should be topic of further research.

Within the discount shopping scenario this technique could lower blocking time if one customer delays to send a "buy" decision because the coordinator can decide for time-out and inform all the customers about aborting the transaction. On the other hand an individual customer can continue his work (i.e. leave the store) immediately after receiving the coordinator's decision.

If a node participates in a transaction and causes a time-out or is temporary unavailable, it must be informed about the transaction status after the recovery of this node. In a fixed scenario this is done by communication with the coordinator or with another participant of the transaction. In a mobile scenario the coordinator may also be a mobile node, therefore it is possible that the coordinator is not available. To preserve the transaction status, some kind of shared storage must be provided where recovering nodes can pick up the state of transactions they where involved in before their communication channel fails.

The most obvious technique would be to place this data on a central server within a fixed network that can be reached via base stations by every mobile client. As we assume that communication with base stations is also unreliable und sometimes is not possible, other approaches to realize a shared storage which every mobile node can access should be evaluated.

5. Summary and conclusions

We have presented two classes of scenarios with slightly different requirements to atomic commit protocols. While applications that follow scenario 1 are not solvable by atomic commit protocols in the sense that no protocol exists which allows deciding for commit, applications that follow class 2 are solvable by an atomic commit protocol. Because the traditional 2-phase commit protocol is not appropriate for mobile networks due to a high probability of node and link failures, we proposed two actions that improve the performance of the 2-phase commit protocol. First we proposed to define appropriate time-out values during the vote-phase based on the connectivity context and secondly to provide some kind of shared storage for mobile clients saving the state of transactions processed by this client before link failures. The concrete design and implementation of these actions is left for further research.

We consider our contribution to be a useful basis for the development of transactional applications in mobile environments.

References

[BHG87] P.A. Bernstein, V. Hadzilacos, and N. Goodman, “Concurrency Control and Recovery in Database Systems”, *Addison-Wesley*, 1987

[Ch93] Panos K. Chrysanthis. Transaction Processing in Mobile Computing Environment. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 77--83, Princeton, New Jersey, October 1993

[DHB97] Margaret H. Dunham, Abdelsalam Helal, and Santosh Balakrishnan. A mobile transaction model that captures both the data and movement behavior. *ACM/Baltzer Journal on Special Topicsin Mobile Networks and Applications*(MONET), 1997.

[FG04] Fife, L. and L. Gruenwald, "TriM: Tri-Modal Data Communication in Mobile Ad-Hoc Networks ", DEXA 2004

[Gr78] J. N. Gray. Notes on data base operating systems. In *R. Bayer, R. M. Graham, and G. Seegmüller, editors, Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, chapter 3.F, page 465. Springer-Verlag, New York, 1978

[HRG00] Haritsa, J. R., Ramamritham, K., Gupta, R.: The PROMPT real-time commit protocol. IEEE Transaction on Parallel and Distributed Systems, 11(2): pp.160-180, 2000.

[Ku02] V. Kumar, N. Prabhu, M. Dunham, Y.A. Seydim, "TCOT - A Timeout-based Mobile Transaction Commitment Protocol", IEEE Trans. on Computers, Vol. 51, No. 10, 2002

[Li02] Yunsheng Liu, GuoQiong Liao, Guohui Li, and JiaLi Xia. Lynch. Relaxed Atomic Commit for Real-Time Transactions in Mobile Computing Environment. In *Advances in Web-Age Information Management, Third International Conference, WAIM 2002*, pages 397-408, Beijing, China, August 11-13, 2002.

[LKS91] Levy, E.,Korth, H. F., Silberschatz, A.: An optimistic commit protocol for distributed transaction management. In: Proceedings of ACM SIGMOD 1991 International Conference on Management of Data, Denver Colorado, pp.88-97, 1991.

[Ly94] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. Atomic Transactions. *Morgan Kaufmann Publishers*, 1994

[Sk81] D. Skeen, Non-blocking commit protocols. In Proceeding of ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, pp.133-142, June 1981.

[Ty96] J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 8-26, May 1996.

[UG01] Hector Garcia-Molina, Jeffrey D. Ullmann and Jennifer Widom, Database Systems: The Complete Book, *Prentice Hall PTR*, 2001

[VL92] George Varghese and Nancy A. Lynch. A tradeoff between safety and liveness for randomized coordinated attack protocols. In *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing*, pages 241-250, Vancouver, British Columbia, Canada, August 1992

[WC99] Gary D. Walborn and Panos K. Chrysanthis. Transaction processing in pro-motion. In Proceedings of the 1999 ACM symposium on Applied computing, pages 389-398. ACM Press, 1999