Making Distributed Transactions Resilient to Intermittent Network Connections

Nuno Santos and Paulo Ferreira
INESC-ID/IST
Distributed Systems Group
Rua Alves Redol nº 9, 1000-029 Lisboa
[nuno.santos, paulo.ferreira]@inesc-id.pt

Abstract

Advances in technology enabled new types of networks to appear (e.g. PDA based spontaneous networks). Here, execution of transactions manipulating distributed objects is affected by the intermittent connectivity thereby causing unnecessary aborts.

This paper presents a solution to make transactions resilient to intermittent connections thus increasing transaction throughput. This is achieved by i) allowing transactions to further span in time and/or ii) relaxing the consistency properties of transactions. For this purpose, application programmers specify the minimum transaction requirements (which depend on the application semantics) using policies. Evaluation shows that slightly increasing the maximum transaction execution time and/or reducing consistency, transaction throughput increases substantially.

This solution is implemented in MobileTrans – a distributed object-oriented middleware system providing adaptive transactions aiding the reliable management of distributed object graphs. Policies are specifically designed to overcome the connectivity intermittence problem. The minimum transaction requirements are specified as parameters to these policies without having to change application code.

1. Introduction

Advances in technology enabled the appearance of portable devices, such as laptops, cell phones and PDAs, equipped with wireless interfaces (eg. Bluetooth and Wi-Fi) allowing them to communicate and share data with fixed stations or with other portable devices. Spontaneous networks form when portable devices are in a range allowing the establishment of wireless links between them. These networks are, by nature, dynamic; nodes enter and leave networks, communication links are more prone to failures and disconnections are frequent. This feature leads distributed transactions running on these environments to abort unnecessarily due to connectivity problems even though conflicts do not exist between transactions. Current transaction systems [6, 9] provide limited adaptability to deal with variable network conditions and applications semantics.

This paper presents a solution for making transactions resilient to intermittent connectivity. When connections fail, transactions are prevented from aborting by i) postponing transactions until unavailable nodes become available and/or ii) by relaxing the ACID[1] [1] properties (e.g. providing inconsistent object replicas or discarding some updates performed by transactions). The rationale is that applications may trade some properties offered by the strict transactional model for a higher commit rate. Evaluation shows that, slightly increasing the transaction execution time and/or reducing acidity, increases the transaction commit rate substantially.

These mechanisms are strongly dependent on the application semantics since their side effects (spanned transaction execution time due to transaction postponing and consistency degradation due to acidity relaxation) may be unsafe. Thus, application awareness is provided; application programmers setup the *execution mode* and the *minimum requirements*. There are two execution modes – *min-time* and *max-consistency* modes – chosen whether faster transactions are preferred (with possible loss in data consistency) or whether consistent transactions are preferred (with potentially high execution times), respectively. The minimum requirements characterize the conditions below which transactions are aborted: the maximum execution time and the minimum consistency level. Several *consistency levels* are defined for relaxing consistency.

Intermittent connectivity adaptation is achieved using MobileTrans [8]. It is a transactional distributed object-oriented middleware system providing transactions with an adaptive behavior to mobile environments. Transaction behavior is specified through declarative policies; it is not necessary to change the application code. A built-in system policy enforces the previously described adaptive behavior to intermittent connections. Application programmers pa-

[1] Atomicity, consistency, isolation and durability

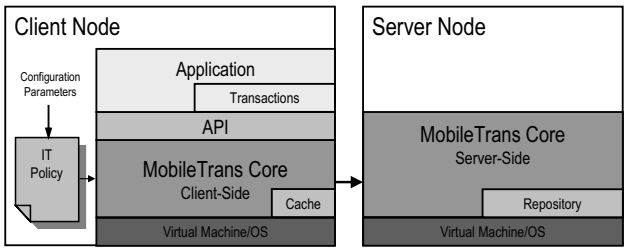

Figure 1. Architecture of MobileTrans.

rameterize this policy by providing the *execution mode* and the *minimum requirements*; no complicated policies have to be developed and configuration is intuitive.

This paper is organized as follows. Section 2 briefly describes the MobileTrans system. Section 3 presents how transactions are handled in the presence of intermittent connections. Section 4 presents and discusses the obtained experimental results. Section 5 surveys the relevant related work. Finally, Section 6 draws the conclusions of this paper and presents future directions.

2. Architecture

MobileTrans is an adaptive transactional object-oriented middleware system supporting applications for dynamic environments. This system provides *transaction policy* support; application programmers provide declarative statements (transaction policies) evaluated at run-time that adapt the transactions behavior according to the changes of the environment and the application semantics.

The MobileTrans architecture (see Figure 1) is client-server. Both client and server run on top of a virtual machine. The MobileTrans server stores and exports shared data. MobileTrans clients are applications bundled with a client-side core component enabling applications to access the data provided by the servers (running locally or in other nodes). Nodes may be providers and consumers of data if executing both server and client code.

Data is represented as *object graphs*. Objects are stored in *repositories* managed by the MobileTrans servers. There is a single consistent version of each object, stored in a repository whose node is called the object *home node*; clients cache replicas of object (sub)graphs accessed by local client applications in a client-side *cache*. It is not guaranteed that objects fetched from caches are consistent.

Transactions are issued by client applications and follow a distributed transaction model. Transactions start with the *begin* operation and end upon issuing the *commit* or the *abort* operations. Objects are created and accessed within transactions. To access remote objects, objects are first replicated from a source site (the object home node, the local cache or other client cache); this operation is called *object fetching*. After fetching, accesses are performed on the local object replicas (i.e. by invoking object methods or manipulating object fields). Transactions finish successfully (i.e. all updates performed locally are made persistent at their home nodes) if the commit operation succeeds.

Transactions behavior is independently specified in a XML declarative file called *transaction policy*. The policy mechanism and the mobile-aware transaction protocol support incorporate several configurable features [8] that affect the properties of transactions and allow the fine-grain tuning of transactions behavior. To deal with the intermittent connectivity problem, a specific system provided intermittent transaction policy (IT policy) was developed (see Figure 1). This policy is configurable through parameters provided by the application programmers (see Section 3).

3. Intermittent connectivity resilience

If connections are intermittent, nodes often cease to be available. Normally, if transactions perform an operation requiring accessing a node that is not available (object fetching and commit operations), transactions would abort. To prevent unnecessary aborts, a recovery procedure f_T is applied to the failing operation of transaction T. There are two possible side effects to T resulting of applying f_T: an increased transaction execution time or/and a loss of data consistency. The *transaction tolerance time* (t_T) and *transaction consistency level* (c_T) indicators are defined to control these effects on transactions.

Tolerance time $t_T = max(t_{f_0}, \ldots, t_{t_n}, t_c)$, where t_k stands for the time that the transaction operation k requires to wait for it to succeed; $k = f_i$ stands for the i^{th} fetch operation and $k = c$ for the commit operation. Informally, the tolerance time provides the maximum time that any network dependent operation performed by the transaction has to wait until that operation is able to proceed successfully; e.g., if connections are stable during the transaction execution, then $t_T = 0s$.

Consistency levels are specified in Table 1. Each level characterizes the requirements that transactions must hold both at fetching time (to read objects) and at commit time (to submit the object updates). Regarding fetching, it is possible to: i) get a full consistent replica of the required object; ii) get any replica (possibly inconsistent) of the object; iii) provide a null replica.[2] Thus, in the first case, a specific node must be available (the objects home node); in the second case, any node containing a cached object replica (even the local cache) must be available; in the third case, no node has to be available for the operation to succeed. Regarding commit, it is possible: i) to validate all transaction updates which requires that all participants be available, ii) to discard the updates whose nodes are not available at commit

[2] Obviously, application programmers must be aware of the possibility of being provided inconsistent replicas or even null replicas.

c	Fetching	Commit
5	full consistent replicas	no updates can be dropped
4	full consistent replicas	updates can be dropped
3	inconsistent replicas	no updates can be dropped
2	inconsistent replicas	updates can be dropped
1	null replicas	no updates can be dropped
0	null replicas	updates can be dropped

Table 1. Consistency level (c).

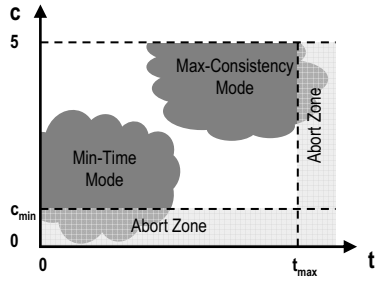

Figure 2. Distribution of the transactions c_T and t_T depending of the applied mode.

time (the updates w.r.t. the absent participants are discarded and the updates w.r.t. the available ones are validated[3]).

In order to minimize aborts in the presence of intermittent connections, the recovery procedure f_T specifies a course of action that may increase t_T, decrease c_T or both. This course of action depends on the *minimum requirements* and the *execution mode* specified by the application programmer. The former declare the values, both in terms of the c_T and t_T, below which transactions are aborted: c_{min} specifies the minimum consistency level and t_{max} specifies the maximum tolerance time. The execution mode (m) characterizes the t_T and c_T balancing method. There are two modes: the *min-time* and the *max-consistency* modes. Which mode is preferable depends on the transaction properties that are mostly important to safeguard from the application programmer's viewpoint. The former privileges faster transactions in detriment of a possible loss of data consistency. The latter privileges high consistency of data allowing longer execution delays. Figure 2 represents the pairs (c_T, t_T) of transactions when executing in the min-time mode or when executing in the max-consistency mode.

In short, to describe how transactions overcome intermittent connections, application programmers specify the tuple $[m, c_{min}, t_{max}]$. The behavior taken by each of the execution modes and the explanation of how t_{max} and c_{min} are handled in each execution mode is presented in the remainder of this section.

[3]MobileTrans allows objects of the transaction to be selectively marked with different levels of consistency.

3.1. Min-time mode

In the *min-time* mode, when a network dependent operation cannot proceed, the transaction is postponed until c_{min} is reached. This may imply reducing the consistency level c_T (as long as $c_T \geq c_{min}$). Once c_{min} is reached, the operation concludes. If the t_{max} is exceeded and the c_{min} is not achieved, the transaction is aborted. Thus, the min-time mode aims at executing transactions as fast as possible, trading it for a possible loss in consistency.

For example, suppose a configuration where $c_{min} = 3$ and $t_{max} = 15s$. If a transaction is fetching an object, but its home node is absent, an inconsistent replica of the object is searched in the local cache or in a cache of a nearby node. If the replica is found, the c_{min} is reached and the transaction proceeds; otherwise, the transaction is postponed for a maximum of 15s waiting that a node carrying the consistent or an inconsistent replica of the object becomes available. If t_{max} exceeds without this to happen, the transaction aborts.

3.2. Max-consistency mode

In the *max-consistency* mode, when a network dependent operation cannot proceed, the transaction is postponed until the consistency level is as high as possible ($c_{max} = 5$). Until the consistency level does not reach c_{max}, the transaction is postponed for a maximum time t_{max}. If t_{max} expires, the transaction is checked for the reached consistency level c_T. If $c_T \geq c_{min}$, the transaction proceeds with consistency c_T; otherwise, it is aborted. Thus, the max-consistency mode aims at executing transactions consistently, spanning, if necessary, for longer periods of time.

For example, suppose that $c_{min} = 1$, $t_{max} = 3600$ and a transaction is trying to fetch an object from its absent home. The transaction is then postponed until its home node becomes available or t_{max} exceeds. If t_{max} (1 hour) exceeds, the maximum consistency level c achieved at that time is applied (if $c_T \geq c_{min}$); for example, if, in the meantime, an inconsistent replica of the object was found in the local cache or in the cache of a nearby node, $c_T = 3$; if no replica was found during that period, $c_T = 1$ (i.e. the c_{min}).

4. Evaluation

The MobileTrans prototype was implemented in Microsoft .Net and the intermittent transaction policy written in a MobileTrans XML-based language. To evaluate the proposed solution, several micro-benchmarks were conducted. These consist of test transactions executed during a fixed simulation time. Half the transactions are read-only while the other half is read-write. For each simulation, ~ 130 transactions are randomly launched. The total simulation time is 36s. Each transaction accesses 100 objects, 64 byte each. For simulation purposes, micro-benchmarks

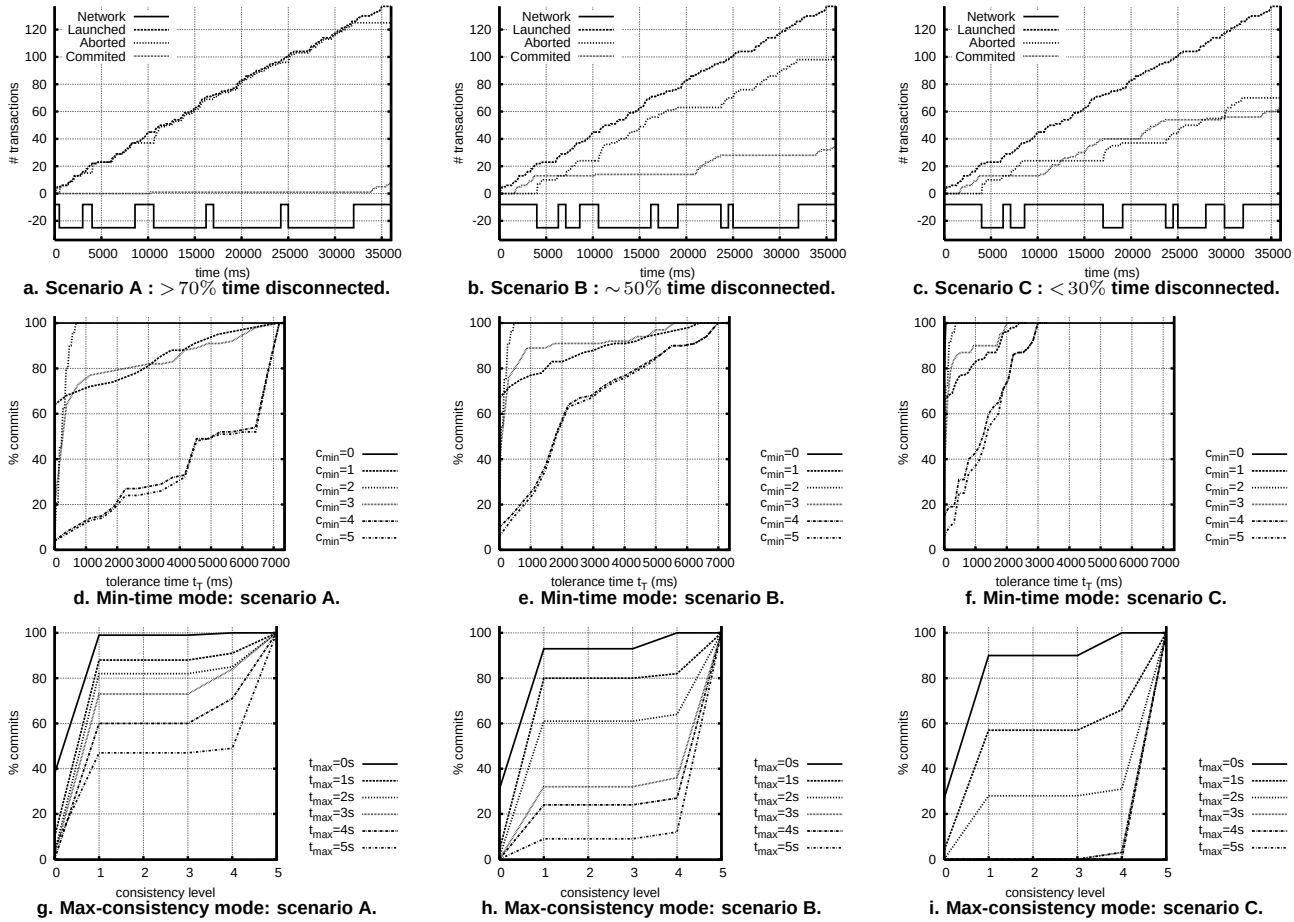

Figure 3. Evaluation results for three simulated environments.

were executed on two Pentium 4, 2.8 Ghz, 512 MB PCs connected by a 100Mb LAN: one PC running the MobileTrans server, the other PC executing the console micro-benchmark applications acting as clients.

Three simplified connectivity patterns were used in the simulations. These patterns attempt to modulate real world scenarios (see Figures 3.a-c). The square wave in the basis of each Figure represents the connectivity pattern: if it is high (low), there is (not) connectivity with the destination node. Scenario A (Figure 3.a) refers to a pattern where the total disconnection time is high ($> 70\%$ the simulation time). Such situation is common among nodes that seldom meet, but meet periodically and perform cooperative work, e.g. workers of a transports company. On the contrary, in scenario C (Figure 3.c) the total disconnection time is reduced ($< 30\%$ the simulation time). An example is the typical work meeting where connections may be interrupted temporarily because one leaves the room for a coffee break. Scenario B (Figure 3.b) provides an intermediate situation where disconnections occupy $\sim 50\%$ the simulation time. For each scenario, Figures 3.a-c describe the evolution of transactions in time – the total number of transactions that are launched, committed and aborted due to the intermittent connectivity – without applying the recovery procedure f_T. Results show that the number of aborting transactions is very high. Even in the highest connectivity scenario (scenario C), $\sim 50\%$ of transactions aborted.

Simulations were performed separately for both execution modes. To observe the impact of t_{max} and c_{min} in the transaction throughput, the minimum requirements were varied and properly calibrated so that all transactions could commit. The series of Figures 3.d-f (Figures 3.g-i) show the results achieved for the min-time mode (max-consistency mode). These results only refer to transactions that would abort if recovery was not applied. The remainder of this section analyzes these results and draws conclusions.

4.1. Min-time mode evaluation

The relevant factor affecting the transaction tolerance time is the c_{min}; the less demanding are the consistency requirements, the sooner conditions are achieved and trans-

actions may proceed. The t_{max} bound is the factor that may drive transactions to abort since, when t_{max} expires, if c_{min} was not achieved, transactions are aborted. Figures 3.d-f reflect how transaction throughput changes for several c_{min}. These results were obtained with a t_{max} equal to the simulation time such that all transactions commit. Thus, it is possible to determine the abort rate for lower t_{max}, e.g. (see Figure 3.d, $c_{min} = 5$) setting $t_{max} = 2s$, the commit rate is $\sim 20\%$. From these results two observations can be drawn: i) by admitting a slight reduction of the consistency level ($c_{min} = 3$) transaction throughput increases substantially within a small period of time – by fixing $t_{max} = 0.4s$, $\sim 70\%$ of the aborting transactions are now able to commit, ii) if high consistency levels are needed ($c_{min} \geq 3$), all transactions are able to commit if setting t_{max} to the maximum disconnection period.

4.2. Max-consistency mode evaluation

In the max-consistency mode, transactions execute with the highest possible consistency level. Thus, the relevant factor affecting the achieved consistency level is t_{max}; e.g. if $t_{max} = \infty$, transactions always achieve the maximum consistency level (assuming that eventually nodes become reachable); however, transactions may last indefinitely. The aborting factor is c_{min} since, when t_{max} expires, if it is not possible to enforce the c_{min} at that time, transactions are aborted. Figures 3.g-i reflect the transaction throughput for several t_{max}. Results show that as connectivity time increases, the number of transactions that are able to commit with the maximum consistency increases substantially. In Figure 3.i, all transactions achieve $q_T = 5$ with $t_T \leq 5s$.

5. Related Work

The Rover toolkit [2] is a client-server distributed object model designed for mobile networks. It allows application programmers to build their own transaction models. However, this is a complex and cumbersome task since consistency enforcement is not separated from the application code. Proposals such as Clustering [5] and Prewrite [4] support mobile transactions by relaxing the strict ACID model. These are mainly focused on the operation in disconnected mode other than situations of intermittent connectivity. Mobisnap [6] is a database middleware system that supports applications running on mobile environments. It allows caching of relational data in the clients who concurrently update the database. The use of reservations provides some support for conflict avoidance and reconciliation but it lacks support for specifying the behavior of transactions. In ProMotion [7] and Moflex [3] it is possible to specify how mobile transactions behave during handover or how to perform hoarding and reconciliation of data. SyD [9] provides services for performing QoS-aware transaction processing across multiple devices of a mobile network. It is similar to MobileTrans w.r.t. providing atomicity and consistency degrees and policies for tuning them. However it is not clear how SyD is effective handling transaction execution under intermittent network connections.

6. Conclusions and Future Work

In this paper we present a solution for making transactions resilient to intermittent connections. An increased transaction throughput is achieved by allowing transactions to delay and by relaxing the consistency properties of transactions. This solution is built on top of the MobileTrans system which provides a policy based flexible transaction management for mobile environments. Evaluation results are encouraging since slightly increasing the maximum transaction execution time and/or reducing consistency, transaction throughput increases substantially.

As future directions, we underline the following. To automatically setup the optimal c_{min} and t_{max} by feeding back the connectivity patterns sensed by the network, to study further mobility scenarios and to appropriately describe them through policies, to improve the consistency model so that it becomes more intuitive to the application programmers. In short, to enhance the transaction model by designing new features that increase transaction efficiency and facilitate implementation of transactions.

References

[1] J. Gray. Notes on database operating systems. In *Operating Systems: an Advanced Course. Lecture Notes in Computer Science*, volume 60, pages 394–481. Springer-Verlag, 1978.

[2] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek. Mobile computing with the rover toolkit. *IEEE Transactions on Computers*, 46(3):337–352, 1997.

[3] K.-I. Ku and Y.-S. Kim. Moflex transaction model for mobile heterogeneous multidatabase systems. In *Procs. of the 10th IWRI in Data Engineering*, San Diego, 2000.

[4] S. K. Madria and B. Bhargava. A transaction model for improving data, availability in mobile computing. *Dist. and Parallel Databases: An Intl. Journal*, 10(2):127–160, 2001.

[5] E. Pitoura and B. Bhargava. Maintaining consistency of data in mobile distributed environments. In *Procs. of 15th ICDCS*, Vancouver, Canada, 1995.

[6] N. M. Preguica, C. Baquero, F. Moura, J. L. Martins, R. Oliveira, H. J. L. Domingos, J. O. Pereira, and S. Duarte. Mobile transaction management in Mobisnap. In *ADBIS-DASFAA*, pages 379–386, 2000.

[7] K. Ramamritham and P. K. Chrysanthis. A taxonomy of correctness criterion in database applications. *Journal of Very Large Databases*, 4(1), 1996.

[8] N. Santos, L. Veiga, and P. Ferreira. Transactions policies for mobile networks. In *Procs. of the 5th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, 2004.

[9] W. Xie, S. B. Navathe, and S. K. Prasad. Supporting QoS-aware transactions in a system on mobile devices (SyD). In *Proc. of the 23rd ICDCSW*, 2003.