E-MEDISYS 2008
2nd International Conference: **E-M**edical **S**ystems
October 29-31, 2008 – TUNISIA

Implementation and Evaluation of Delay-Aware and Fault-Tolerant Mobile Transactions

Brahim Ayari, Abdelmajid Khelil, Neeraj Suri and Eugen Bleim

Technische Universität Darmstadt,
Dependable Embedded Systems and Software Group,
Hochschulstr. 10, 64289 Darmstadt, Germany

brahim@informatik.tu-darmstadt.de
khelil@informatik.tu-darmstadt.de
suri@informatik.tu-darmstadt.de
jonnybleim@gmx.de

Abstract: Pervasive healthcare is an emerging discipline, where mobile embedded systems are increasingly used to support transactions. Such systems entail a range of heterogeneous entities - both the embedded devices and the networks connecting them. While these systems are exposed to frequent and varied perturbations, the support of atomic distributed transactions is still a fundamental requirement to achieve consistent decisions. Guaranteeing atomicity and high performance in traditional fixed wired networks is based on the assumption that faults like node and link failures occur rarely. This assumption is not supported in current and future mobile healthcare embedded systems where the heterogeneity and mobility often result in link and node failures as a dominant operational scenario. In this paper we summarize our work to provide for atomic commit protocols for mobile environments where consistency can not be compromised, for example healthcare systems. We present the implementation and experimental evaluation of a commit protocol showing its suitability for healthcare environments.
Key words: Transactions, mobile database systems, dependability, healthcare systems.

INTRODUCTION

Future mobile medicine and healthcare home pervasive computing scenarios aim to make healthcare available to anyone, anytime and anywhere [MOH]. The use of wireless technologies and mobile devices allows for pervasive access to health care services. The corresponding computing environments are increasingly characterized by frequent and varied perturbations. These are directly apparent to the delivery of services as constraints and failures. These mobile systems are also constrained by the scarcity of processing, storage and energy resources of mobile devices, and the continuously varying properties of wireless channels. Most of the failures which can occur in such systems are caused by node (given the mobility and size of these nodes) or communication failures. These failures can last from seconds, minutes to even hours, e.g., network partitioning. Also increasingly, the mobile medicine and healthcare environments involve applications such as life assist systems, patient data management, body area networks [JON 01] that require strict atomic database transactions guaranteeing strict data consistency. Atomic commit protocols ensure strict atomicity of database transactions and play therefore a major role for the design of these applications. In the literature computer transactions are usually considered to be delay-sensitive. Accordingly, most of existing atomic protocols display very limited perturbation-tolerance leading to either poor transaction commit rate or to high resource blocking time which consequently decreases the efficiency of the mobile healthcare system. In our previous work [AYA 06], we showed that sacrificing latency (time needed to decide about the outcome of the transaction) is necessary to cope with frequent and enduring perturbations without sacrificing performance in terms of efficiency and commit success rate.

In mobile healthcare systems both delay-sensitive transactions (for data with low time validity) and delay-tolerant transactions (for data with high time validity) are needed. A limited body of research exists for real-time transactions [HAR 00] [LIU 02]. However, to the best of our knowledge, delay-aware transactions have not yet been addressed. In [AYA 08]

Research supported in part by EC CoMiFin, NoE ReSIST and DFG GRK 1362 (TUD GKMM)

we argued for the necessity of delay-awareness of mobile transactions in networked embedded systems. Our work in [AYA 06] investigated primarily infrastructure-based system models. We extended this base model in [AYA 08] to cope with a more generalized mobile system that also involves ad-hoc communication scenarios. In this paper we build upon our prior work to (a) detail delay-aware transaction handling for environments such as mobile healthcare, and (b) provide a real implementation.

The remainder of this paper is organized as follows. In Section 2, the system model is described along with a classification of perturbations in mobile healthcare environments. A mobile transaction commit protocol, referred to as FT-PPTC, is presented in Section 3. In Section 4, the implementation of the presented protocol is discussed. Section 5 presents experimental measurements expressing the performance of the FT-PPTC protocol. Section 6 concludes the paper and briefly outlines the future work.

1. Related Work

Mobile transactions are increasingly the focus of extensive ongoing research [DUN 97] [CHR 93] [PIT 95] [MAD 01], and some recent commit protocols have also been proposed in the literature [BOB 00] [KUM 02] [NOU 05].

Unilateral Commit for Mobile *(UCM)* [BOB 00] supports disconnections and off-line executions on mobile devices. UCM is a one-phase protocol where the voting phase of the two-phase commit (2PC) [GRA 78] is eliminated. The coordinator acts as a “dictator” imposing its decision on all participants. UCM guarantees atomicity. However due to its strict assumptions (strict two-phase locking [BER 87] required for all participants), the data accessed by uncommitted local transactions remain locked until the whole transaction is committed or aborted. In addition, the UCM assumes homogeneous database systems that is often not viable in mobile environments. Transaction Commit On Timeout (TCOT) [KUM 02] uses timeouts to provide a non-blocking protocol that limits the amount of communication between the participants in the execution of the protocol. Instead of exchanging messages to reach a Commit or Abort decision, the coordinator waits for timeouts to expire. TCOT provides only semantic atomicity as defined in [GAR 83], which is weaker than the desired strict atomicity [HAE 94] for transactions. This limits the applicability of TCOT. Furthermore, TCOT does not consider mobile hosts as active participants in the execution of transactions.

Mobile two Phase Commit (*M-2PC*) [NOU 05] considers mobile hosts as active participants. A mobile participant executes its fragments and delegates the commitment to its agent on a fixed host. Unfortunately, M-2PC assumes that all mobile participants are connected at the transaction initiation and that network disconnections are allowed only after the mobile host delegates its commitment duties.

A common drawback of these protocols is that they address only a small subset of the numerous failure types that may occur in the mobile environment. Furthermore, these protocols are not robust to coordinator failures. In our work [AYA 06] [AYA 08], we propose a comprehensive fault model and design our protocol to tolerate faults resulting in enhanced transaction resilience compared to existing protocols.

2. System and Fault Models

We first present a model of the mobile healthcare (MHC) environment. Consequently, we elaborate the system model, and the corresponding transaction and fault models.

2.1. Model of the Mobile Healthcare Environment

We consider a mobile distributed environment consisting of a set of mobile hosts (MH) and a set of fixed hosts (FH). The MHs intermittently connect to the wired network through Mobile Support Stations (MSS) via wireless channels (Figure 1). MHs can communicate with each other or with fixed entities using only the services provided by the MSSs. We refer to the set of MHs as $M = \{MH_1, \dots , MH_m\}$, and that of FHs as $S = \{FH_1, \dots , FH_s\}$, where m and s are the number of MHs and FHs respectively.

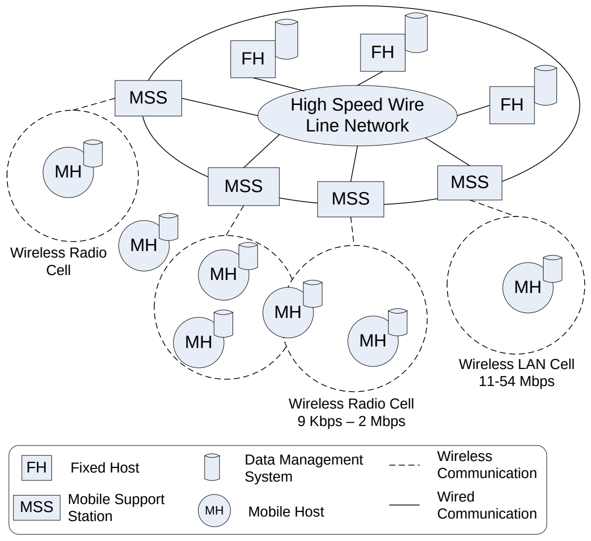


Figure 1. *Architecture of MHC environments*

We assume that each MH has a Mobile Database Server (MDBS) installed on it, and that a Database Server (DBS) is attached to each FH. Database servers are needed on both fixed and mobile hosts to support basic transaction operations such as read, write, commit and abort.

2.2. System Model

We consider applications, which run on either mobile or fixed hosts and access data located on both *mobile* and *fixed* hosts. A transaction can originate from any host in $M \cup S$, and the participants in its execution can be any set $P \subseteq M \cup S$. However, most of mobile transactions involve some FHs as participants.

The hosts in the considered mobile environment may entail different hardware and software implementations, including their *database management systems*. Thus, we are dealing with *heterogeneous* mobile databases. MHs are also heterogeneous and can range from smart cards, mobile phones and personal digital assistants (PDAs) with restricted storage and processing capabilities to laptops with considerably higher capabilities.

We consider the distributed database system components ($\in M \cup S$) to be *autonomous*. Autonomy means that the components of the system are able to perform their tasks independently from each other. With respect to the execution of transactions, this requires that every component must take the decision to commit or abort the transaction independently from other components in the network. Components are also able to decide which information to share with the global system and how to manage their own data. The data of the MH is replicated on a fixed backup database server.

We assume the existence of a *coordinator (CO)*, which is responsible for coordinating the execution of the corresponding transaction. For different transactions, different nodes may play the CO role. The CO is responsible for storing information concerning the state of the transaction execution. Based on the information collected from the participants of the transaction, the CO takes the decision to commit or abort the transaction and informs all participants about its decision.

The MH is not considered to have a physical stable storage, as it can be subject to loss, damage and random disconnections. For this reason the MH is not desired to take on the CO role. We assume only FHs to have stable storage. Thus, the CO role should be performed by one or more FHs. If a MSS has similar capabilities to that of a FH, the CO role can also be performed by a MSS. The CO maintains information about the connectivity of the MHs participating in the execution of the transaction (i.e., whether they are connected to the network or not).

We do not place any restrictions on the storage capabilities of FHs. Further we assume that all DBSs attached to the FHs support the *prepare* operation of the 2PC protocol as a basic operation.

2.3. Transaction Model

Users (doctors, nurses, patients etc.) issue transactions from MHs. After processing a transaction, the system provides the result of the transaction on the user's MH. The transaction may be entirely executable at the user's MH. More often, the transaction has to be fragmented and distributed among a set of nodes $P \subseteq M \cup S$. We refer to a distributed transaction where at least one MH participates in its execution as a *Mobile Transaction (MT)*. Similar to the concept of transaction formalization presented in [OZS 91], we formally define the MT T_i as a triple $<F_i, L_i, FLM_i>$, where $F_i = \{e_{i1}, e_{i2}, \dots, e_{in}\}$ is a set of n "execution fragments" [KUM 98] [KUM 00], $L_i = \{l_{i1}, l_{i2}, \dots, l_{ik}\}$ is a set of k locations in $M \cup S$ $(k \leq m + s)$, and $FLM_i = \{flm_{i1}, flm_{i2}, \dots, flm_{in}\}$ is a set of fragment-location mappings (flm's), where $\forall j, flm_{ij}(e_{ij}) = l_{ij}$, $1 \leq j \leq k$. Although the execution fragments of T_i are semantically related, each one of them can commit independently given the autonomy of their corresponding locations, leading to the commit of T_i.

We refer to the MH, where T_i is initiated, as Home MH (H-MH). The commit set consists of all FHs and MHs participating in execution and commit of T_i including H-MH. FHs and MHs in the commit set are called participant FHs (Part-FH) and participant MHs (Part-MH) respectively.

The database system installed on MHs provides backup facilities to assist with the recovery of the database. To achieve a complete recovery, all the operations completed on the MH need to be stored on a stable and reliable storage on a FH. Thus, the MHs are able to take part in the execution of transactions in the considered environment independently of other system components.

2.4. Fault Model

Designing a fault-tolerant transaction commit protocol essentially requires the identification of constraints and failure modes that can occur in the considered environment. The following sections enumerate these aspects.

2.4.1. *Constraints*

The MHC environment is constrained mainly by the characteristics of *MHs* and *wireless links*. *MHs* intuitively possess less computational resources such as processor capabilities and storage capacity than the FHs. This increases the time MHs need to execute transactions or may even lead to execution failure. Furthermore, MHs are highly vulnerable to physical loss or damage, and may run in different energy modes or be put-off to save energy. Therefore, MHs naturally show frequent and random network disconnections.

The effective bandwidth available for the MH over a *wireless link* is highly dynamic. This depends on the wireless technology, access coverage, and number of MHs that have to share the medium. These characteristics lead to an unreliable and intermittent network connectivity of MHs.

The limitations and characteristics listed above outline the variation of constraints for the MHC environment being different from those in fixed environments. These constraints also make it harder to design appropriate and efficient commit protocols. A protocol that aborts the transaction, each time the MH disconnects from the network, is not suitable for mobile environments since frequent disconnections are not exceptional but are rather part of the normal mode of mobile operations. Therefore, disconnections need to be explicitly tolerated by the protocol.

2.4.2. *Failure Modes*

We now outline the considered failure modes classified into primary classes of communication and node failures.

Communication Failures: These constitute the most frequent failures in the mobile environment. We distinguish between three kinds of communication failures: *Message loss, communication delay* and *network disconnection*.

Node Failures: We distinguish between MH, FH and CO failures. We separate CO failures from FH failures given the central role CO plays in commit protocols. For MHs, we classify the failures into *transient* and *permanent* failures.

- *Transient MH failures* occur from either software or hardware faults and usually disappear if the MH is restarted. Transient failures are the most probable failures of MHs in the MHC environment. In the case of a transient MH failure, the content of the volatile storage of the MH and consequently the state of its recent computations is lost.
- *Permanent MH failures* are irreparable failures such as the loss or damage of the MH itself or its nonvolatile storage, where the data and logs are stored. Consequently, all the data stored in the MH is lost.
- *CO failures:* We assume a crash-recovery model, i.e., if the CO crashes it stops receiving, sending and processing messages until it recovers after a finite amount of time. Volatile storage of the CO is checkpointed periodically to stable storage and the CO logs its computations and received/sent messages between two checkpoints. Once a backup is done the log is deleted and a fresh logging process is initiated.
- *FH failures:* We assume a crash-recovery model but limit its details as our focus is on failures that are specific to the described MHC environment.

3. The FT-PPTC Commit Protocol

A solution referred by Fault-Tolerant Pre-Phase Transaction Commit (FT-PPTC) for the mentioned commit problem has been presented in [AYA 06]. In this section we shortly review the FT-PPTC protocol before presenting an implementation of it. As key drivers, the FT-PPTC commit protocol ensures the atomicity property. FT-PPTC efficiently minimizes the number of transaction aborts by tolerating the failures described in the fault model, and sacrificing the transaction execution delay. High efficiency is reflected by a low message complexity, especially for wireless messages. Since some FHs participate in most of the transactions, FT-PPTC reduces the sensitive blocking time of resources at Part-FHs.

As Part-MHs may need an arbitrary long time to execute their fragments, and as very few assumptions can be made regarding the connection intervals of MHs, resources of Part-FHs may potentially be blocked for an undefined period of time. Therefore, FT-PPTC *decouples* the commit of mobile participants from that of fixed participants. FT-PPTC splits transaction execution into *two phases*. The first phase, called the *pre-commit phase* (Figure 2), collects “sufficient” information from mobile participants and reduces the commit set to a set of entities in the fixed network. In the second phase the commit involves only FHs and thus can be completed by any atomic commit protocol for wired networks, such as the traditional 2PC protocol [GRA 78]. We refer to the second phase as the *core 2PC phase*.

To allow for this decoupling, we assign a *MH Agent (MH-Ag)* to each Part-MH. The MH-Ag is representing the Part-MH in the fixed network. The MH-Ag is responsible for storing all the information related to the state of all MTs involving the corresponding MH. The MH-Ag is also responsible for executing the 2PC protocol on the behalf of its corresponding Part-MH. The MH-Agent can be implemented by any FH. The CO itself is the MH-Ag of the H-MH.

Intuitively, this decoupling reduces the blocking time of the resources at the fixed devices. It also simplifies the handling of the different types of failures that arise from the mobility of nodes as described in the fault model.

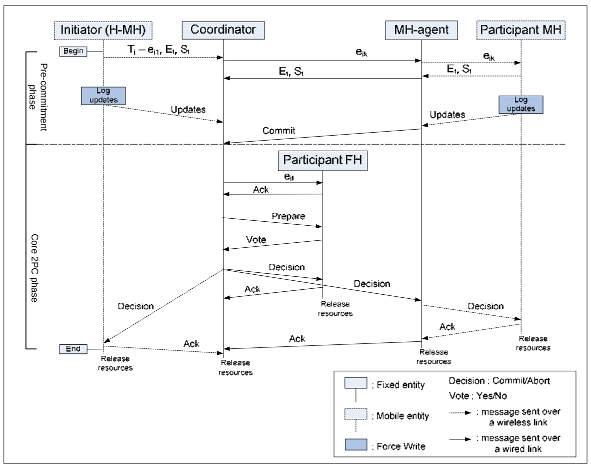


Figure 2. *Failure-free execution of the FT-PPTC protocol*

3.1. Pre-Commit Phase

The pre-commit phase only involves Part-MHs. The MH-Ags act as intermediators between Part-MHs and the CO. Similar to [KUM 02], we exploit a timeout-based concept to reach a provisional Commit decision at the end of the pre-commit phase. Each Part-MH computes an *execution timeout* (E_t), an upper bound for the time to complete the execution of the transaction fragment, and a *shipping timeout* (S_t), an upper bound for the time to compose updates and to send them to the CO. Both timeouts have to account for the constraints of the MH and the wireless link. These timeouts can be extended if needed. Each mobile participant sends both timeout values E_t and S_t to the CO via its MH-Ag.

The CO waits for the expiration of the timeouts of Part-MHs and collects their votes along with the data logs of the H-MH in case of successful execution. The data logs contain the list of all updates made by the MT. The data logs of other Part-MHs (in case of successful execution) are stored by their corresponding MH-Ags. The MH-Ag should be able to propagate the updates made by the Part-MH (in case this has a permanent failure) to its corresponding backup database server using the logs. The CO finalizes the pre-commit phase by a provisional Commit decision or a final Abort decision. The CO decides to proceed to the second phase of FT-PPTC, only if it receives the updates from the H-MH and a "Yes" vote from all MH-Ags (representing the rest of Part-MHs) within the specified time-limit. The transaction is aborted as soon as one Part-MH aborts the transaction or the timeout expires at the CO without receiving either the updates of the H-MH or a "Yes" vote from one of the MH-Ags.

3.2. Core 2PC Phase

As a result of the pre-commit phase, the Part-MHs delegate their corresponding MH-Ags to execute the 2PC protocol on their behalf. The second phase of the protocol begins, when the CO sends the execution fragments of Part-FHs to their corresponding FHs and the 2PC protocol is executed to collect their votes. If all Part-FHs vote for committing the MT, the CO decides to commit it, otherwise it decides to abort. We assume that the 2PC protocol used for collecting the votes from the Part-FHs is modified in such a way that it is non-blocking. This can be achieved using, for example, timeouts to detect message loss.

4. Implementation of FT-PPTC

We first discuss the selection of used technologies and then present the implementation of the FT-PPTC protocol.

For the implementation the object-oriented paradigm is adopted for portability (from one hardware platform to the other) and reusability purposes. JME (Java Platform Micro Edition) was chosen as an implementation platform suitable for so called "embedded consumer products" like sensor nodes, mobile phones, PDAs etc. It represents one of the common accessible solutions for developing applications in mobile environments. The technology allows making the applications reusable and portable between different platforms. JME Platform is a collection of technologies (Figure 3) that can be easily tailored to build a Java environment suitable to the desired goals on the used mobile devices. Figure 3 highlights the modularity of the Java architecture that can be easily customized to our highly heterogeneous mobile healthcare environment. This technology consists basically of three components: (1) a configuration providing basic libraries and a virtual machine (VM), (2) a useful profile as a set of Application Programming Interfaces (APIs) for a selected device, and (3) an optional package with technology-specific APIs. Two configurations are currently provided: the CLDC (Connected Limited Device Configuration), suitable for small/resource-limited devices like mobile phones and the CDC (Connected Device Configuration), appropriate for more powerful devices like PDAs. JVM (Java Virtual Machine) is the interface between the Operating System (OS) and the hardware platform. A VM consists of a class loader, garbage collection and execution engine. It allows compiled applications to be portable and platform independent. VM technology allows also for robust and fault-tolerant implementation as it simplifies fault isolation etc. In our implementation the J9 IBM virtual machine [J9] is used for its advanced features of configurability, compactness and speed. It also provides a predictable architectural layer presenting a common interface for application programs regardless of the underlying device hardware or OS. J9 runs on almost every OS (Windows Mobile, QNX, embedded Linux, OSE, ITRON, etc.) and manages the specific interface with the OS and the hardware of mobile devices.

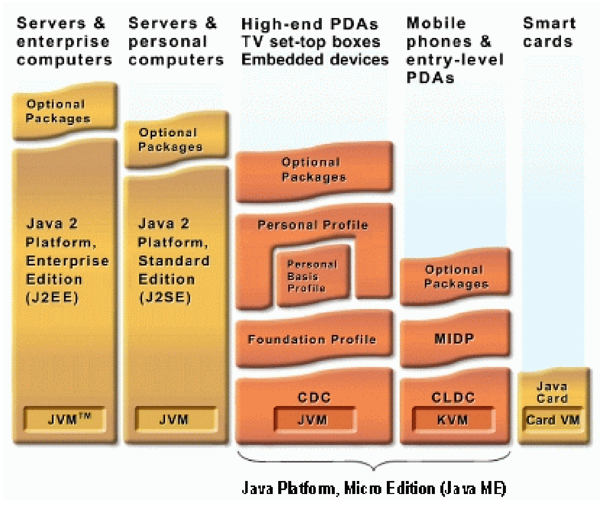

Servers & enterprise computers
Servers & personal computers
High-end PDAs TV set-top boxes Embedded devices
Mobile phones & entry-level PDAs
Smart cards
Optional Packages
Optional Packages
Optional Packages
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Standard Edition (J2SE)
Personal Profile
Personal Basis Profile
Foundation Profile
MIDP
CDC
CLDC
Java Card
JVM™
JVM
JVM
KVM
Card VM
Java Platform, Micro Edition (Java ME)

Figure 3. *Java technologies [JVT]*

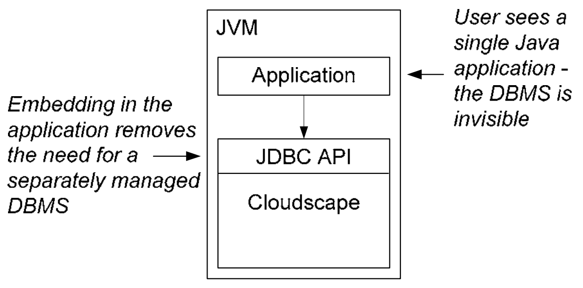

JVM
Application
JDBC API
Cloudscape
User sees a single Java application - the DBMS is invisible
Embedding in the application removes the need for a separately managed DBMS

Figure 4. *Derby embedded in an application [DER]*

For the databases installed on Part-MHs and Part-FHs, *Derby-Cloudscape* or shortly *Derby* [DER] is chosen. Derby is a relational database which is embeddable, non-administrated and Java-coded.

Derby can be embedded in the applications as a Relational Database Management System (RDBMS) engine. By employing Derby in a client application, a Derby system requires no supporting administration (Figure 4).

Optionally Derby can run in Network Server mode as a separate process. This database is strictly implemented in Java. Consequently, Derby is platform-independent and simple to configure and install. Furthermore Derby runs on any certified Java Virtual Machine and needs no proprietary platform environments. This lightweight database engine has a memory footprint of about 2 MB and requires as little as 4 MB Java heap. Derby uses JDBC and provides to write stored procedures and functions in Java without having any proprietary language. Moreover it implements the SQL92E standard [SQL 92] as well as many SQL 99 extensions and includes support of many other RDBMS features like transaction commit and rollback, transactional isolation (serializability), crash recovery, multithreaded connections and online backup.

The core functionality of the FT-PPTC protocol is implemented as Java components which are organized into different packages. This organization allows for an easy customization of the implementation to the target hosts by installing only the required packages. For instance, the MH does not need to implement the services of an agent that are only relevant to a fixed host. Common components which are required by each service build their own package. Figure 5 illustrates the dependencies between the FT-PPTC protocol implementation packages.

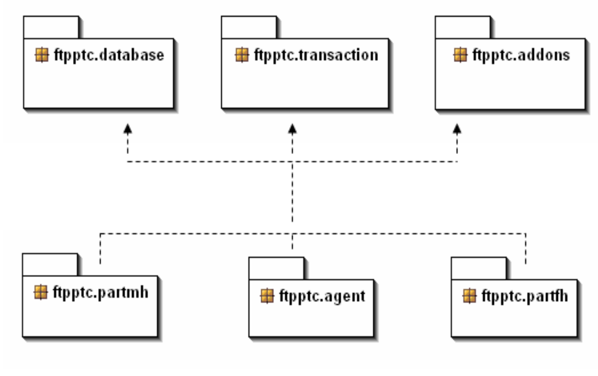
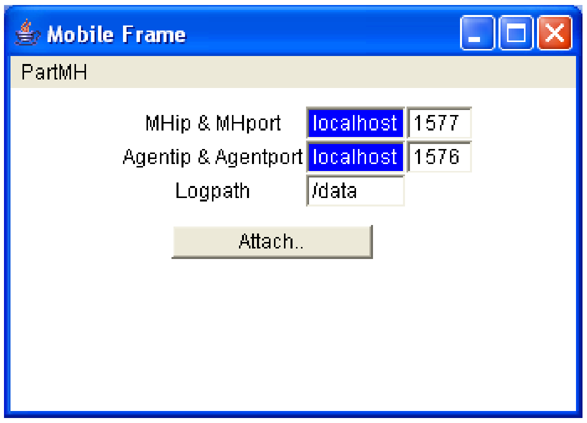


Figure 5. *Packages dependency*

The packages *ftpptc.partmh*, *ftpptc.agent* and *ftpptc.partfh* contain the active components which can be started as a service thread on a corresponding host. The *ftpptc.partmh*, *ftpptc.partfh* and *ftpptc.agent* packages implement the behaviour of Part-MHs, Part-FHs und MH-Ags respectively. There exist no dependencies between these three packages which can be independently deployed to their corresponding hosts. The other packages include only passive components which can be only used by other packages and are either basic or necessary add-on components.

The package *ftpptc.partmh* implements the core functionalities of the FT-PPTC protocol for Part-MHs and the execution of the transaction fragments on them. Additionally, a graphic interface (Figure 6) is implemented to simplify the initiation and configuration of the protocol. It can also be used for debugging purposes.

The *ftpptc.agent* package implements the services offered by the CO and the MH-Ags. The main services offered by this package include the management of transaction and participant states and the administration of log files and backups. This package also contains components responsible for creating the connections between the MH-Ags and their corresponding Part-MHs. After a crash of the MH-Ag, this service first recovers the execution state of the Part-MH by loading the corresponding logs from its stable storage. Based on the loaded state, the agent decides to request needed information from the Part-MH which allows it continuing the execution of the MT.



MHip & MHport localhost 1577
Agentip & Agentport localhost 1576
Logpath /data

Figure 6. *Mobile frame*

The package *ftpptc.database* contains the components that serve to handle all database operations on the local databases. These components allow also the execution of independent sub-services in parallel threads to parallelize the execution and control of MTs. This package is also responsible for starting the database of the corresponding transaction participant either in normal mode or in recovery mode. The recovery mode brings the database to the last consistent state before it crashed.

The package *ftpptc.partfh* implements the core of the 2PC protocol described in [GRA 78].

The *ftpptc.transaction* module is responsible for generating, initiating and executing transactions. Moreover this module records the current state of the execution of a particular transaction. These records are important in case of failures in order to recover and consistently continue the execution of the MT.

The package *ftpptc.addons* offers add-on components to support the modality of the protocol implementation and support its evaluation. This package implements basic functionalities to configure the agents, administrate their communication and measure the throughput.

5. Evaluation

Based on the described implementation in Section 4, we now provide the evaluation of FT-PPTC with respect to throughput, latency and resource blocking time. We first describe the testbed used for our experiments. Subsequently the results of our experiments are discussed.

5.1. Testbed

The equipments used in the testbed are illustrated in Figure 7. Two PDAs with the described configurations in this figure are connected via wireless link to an access point. The access point and the fixed hosts are connected to the same LAN.

Both PDAs, Acer n50 and Fujitsu Loox C550 with the characteristics given in Figure 7, have Derby database installed on them with a few test tables and J9 JVM. The *PartMH* service described in Section 4 is deployed to each of the Part-MHs. This service is started on each Part-MH. The access point provides the wireless coverage for the PDAs and connects them to the fixed network illustrated in Figure 7. All three FHs used in the test environment are standard PCs. The configuration of FHs does not impact the results of our experiments, which depend mainly on our Part-MHs. Two of those FHs are deployed as MH-Ags and the *MHAgent* service implemented by the protocol is started on them. Both FHs are associated to both Part-MHs as their corresponding MH-Ags using the configuration facilities on the FHs and MHs. The third FH is deployed as a Part-FH. The Part-FH starts the *PartFH* service and is consequently ready to participate in any transaction whenever requested.

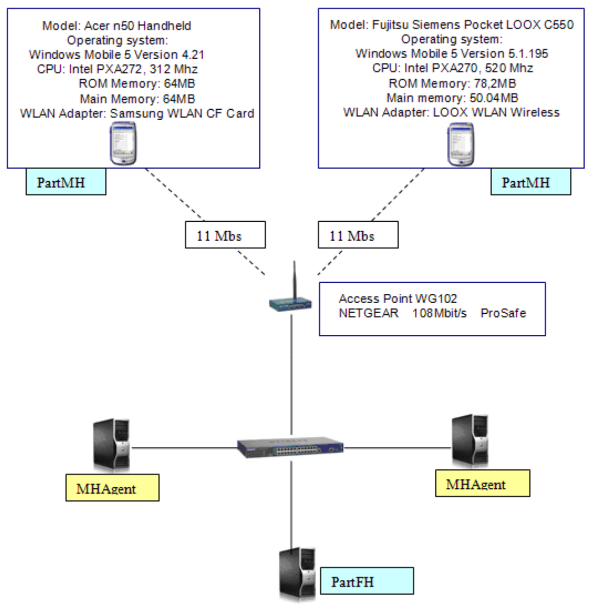

Model: Acer n50 Handheld
Operating system:
Windows Mobile 5 Version 4.21
CPU: Intel PXA272, 312 Mhz
ROM Memory: 64MB
Main Memory: 64MB
WLAN Adapter: Samsung WLAN CF Card
Model: Fujitsu Siemens Pocket LOOX C550
Operating system:
Windows Mobile 5 Version 5.1.195
CPU: Intel PXA270, 520 Mhz
ROM Memory: 78,2MB
Main memory: 50.04MB
WLAN Adapter: LOOX WLAN Wireless
PartMH
PartMH
11 Mbs
11 Mbs
Access Point WG102
NETGEAR 108Mbit/s ProSafe
MHAgent
MHAgent
PartFH

Figure 7. *Testbed*

5.2. Performance

One of the important aspects of protocol evaluation is the investigation of the performance. A commit protocol should maximize transaction commit rate and throughput by minimizing resource blocking time. First we investigate the appropriate timeouts values which should be selected by the Part-MHs in order to minimize the number of aborts due to too short timeouts. Then we investigate throughput and blocking time of FT-PPTC.

5.3. Selection of Timeouts

The value of timeouts is the sum of the execution and the shipping timeout defined in Section 3.1. The optimal timeout varies from one application to another and from one platform to another. Some applications may prefer to achieve higher commit rates by adopting longer timeouts and sacrificing transaction latency. Whereas other applications may select lower timeouts in order to achieve lower transaction latencies leading to poor commit rates. In the following, we investigate the challenging task to find the appropriate timeouts leading to maximal commit rate.

To find out the appropriate timeouts settings, we conduct experiments with different timeout values and also by varying the number of initiated transactions. We start with a timeout equal to zero and increment the value by 100ms for the subsequent experiments. The suitable timeouts for the throughput measurement are the smallest values which yield maximal commit rate.

Figure 8 demonstrates that the commit rate increases with higher values of timeouts for a fixed number of initiated transactions. Each curve represents the percentage of committed transactions for the corresponding value of timeouts and for a fixed number of initiated transactions. The experiments show the existence of a value for the timeouts after which the commit rate is always 100%. This value ranges between 900 and 1000ms for the different experiments conducted.

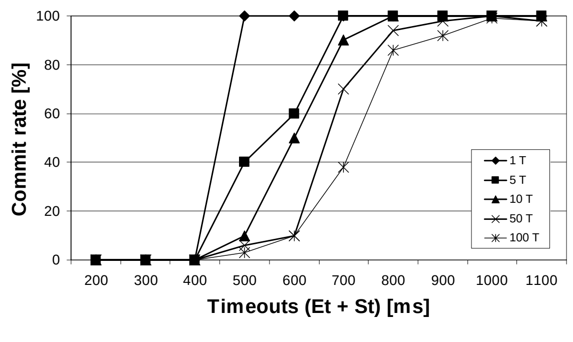


Figure 8. *Timeout selection*

We conducted experiments to collect some intermediate measures on the average times needed to execute different steps of MTs using FT-PPTC in the failure free case. These measurements are presented in Table 1 and present the basis for fixing the experiment settings. A complete transaction that involves two Part-MHs and one Part-FH with one update database operation on each participant may take 2 to 3s. This duration consists of the following protocol execution steps. The execution of a simple database operation like an update statement on the mobile host takes about 200-300ms whereas the protocol part that prepares and finishes the execution block including the execution itself may take up to 550ms. These measures are not valid for the first database operation which lasts even 5 times longer than the subsequent operations (up to 900-1000ms). Therefore the first transaction is not taken into account in order to avoid inaccuracy of the results. The protocol needs 90-150ms to compose the log file. Transmitting the log file over the wireless channel takes 100-200ms. Waiting for receiving the vote of the second Part-MH, executing the core of the 2PC protocol involving the Part-FH, and taking the final decision take from 1-1.5s if each mobile participant voted to commit the MT, otherwise 500-700ms less. The decision handling phase usually takes 100-200ms including committing or rolling back the changes made on the database and sending the acknowledgements.

Protocol step	Execution time on MH
Transaction	2-3 s
Database operation (update)	200-300 ms
Execution with prepare and finish	550 ms
Composing the log file	90-150 ms
Transfer of the log file	100-200 ms
Waiting for decision	1-1.5 s
Decision handling	100-200 ms

Table 1. *Execution time of different protocol steps on MHs*

The measurements of the different protocol steps on fixed hosts deliver different results. Every first execution of an operation on an already started database takes around 100-300ms. Thus, the first transaction is not taken into account while computing the throughput. All the further executions last, on average, between 10 and 30ms. The time needed for composing the update file varies from 20-90ms and the average time to transmit it is usually 10-30ms.

Our experiments showed that the FHs are up to 10 times faster in executing the same database operation than MHs. The transfer of a log file from a MH over wireless connection to a FH is nearly 10 times slower than over a wired connection. The composing of a file is surprisingly approximately 2 times faster on a FH than on a MH.

5.4. Throughput

The transaction throughput is commonly defined as the number of successfully performed transactions per unit of time.

To measure the throughput of mobile transactions it is required to perform transactions involving at least two mobile and one fixed participants. Preferably, the transactions are initiated successively after each other and from the same mobile host to simplify the experiment. The reason behind this configuration is that a simultaneous initiation of transactions involving the same set of participants from different hosts may induce the mutual blocking of resources and falsify the experimental results.

The numbers of transactions initiated in each experiment varies from 0 to 100. For each number of transactions three runs are performed and the average is considered, in order to smooth the impact of outliers. For the timeouts we used the values E_t=300ms for the execution timeout and S_t=700ms for shipping timeout conform to the value of the sum of these timeouts (E_t +S_t = 1000ms) fixed in the previous subsection.

Figure 9 illustrates the throughput of the FT-PPTC protocol which shows a stable throughput as the number of initiated transactions increases. The throughput reaches a maximum and then starts to slightly decrease as maximum load on fixed FH is reached

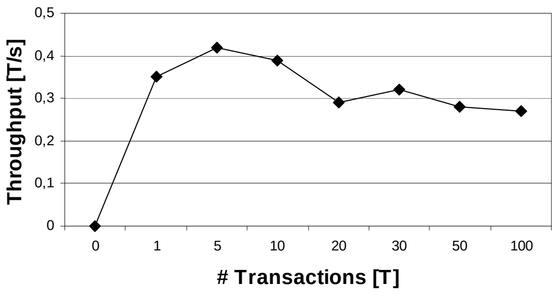


Figure 9. *Transaction throughput*

5.5. Blocking time

The FT-PPTC protocol is primarily designed to provide short blocking time of the valuable resources of FHs. This is achieved by decoupling the execution of Part-MH fragments from Part-FH fragments. To measure the blocking time we continue to use two Part-MHs and one Part-FH similar to the throughput study. Each conducted experiment is also repeated three times under the same conditions and the average is considered.

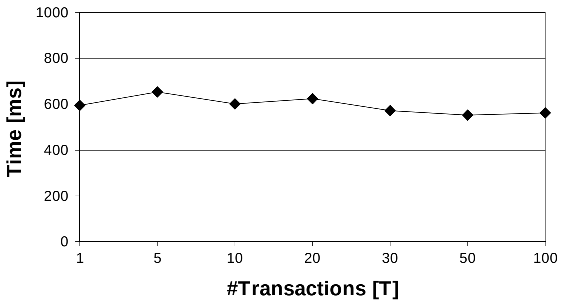


Figure 10. *Blocking time*

Figure 10 shows how the blocking time of the resources on the Part-FH varies with the number of transactions. The results show that this blocking time is relatively constant and does not depend on the number of initiated transactions. The average blocking time varies between 550 and 650ms. In the conducted experiments the minimum blocking time value is 450ms and maximum value is 900ms. These results highlight the scalability of the protocol with respect to the number of initiated transactions and consequently the number of (mobile) participants.

6. Conclusion

In this paper, we first summarized our previous work on developing fault- and delay-aware mobile transactions. We then adopted our main solution, the Fault-Tolerant Pre-phase Commit (FT-PPTC) protocol, to the crucial domain of pervasive healthcare systems. We have further presented a Java implementation of FT-PPTC leading to a real testbed for fault-tolerant and delay-aware mobile transactions. Finally we used this testbed to investigate the performance of FT-PPTC and to give further insights of the protocol under real deployment conditions. The experiments highlight the benefits of sacrificing transaction delay for the purpose of fault-tolerance.

In future, we plan to extend the communication model and design atomic commit protocols for ad-hoc networks which are being increasingly used in pervasive healthcare systems.

REFERENCES

[AYA 06] B. Ayari, A. Khelil, and N. Suri, “FT-PPTC: An Efficient and Fault-Tolerant Commit Protocol for Mobile Environments,” *SRDS’06*, pp. 96–105, 2006.

[AYA 08] B. Ayari, A. Khelil, and N. Suri, “Delay-Aware Mobile Transactions,” 6th *IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS)*, pp. 280–291, 2008.

[BER 87] P. A. Bernstein, et al., *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[BOB 00] C. Bobineau, et al., “A Unilateral Commit Protocol forMobile and Disconnected Computing,” *PDCS*, 2000.

[CHR 93] P. K. Chrysanthis, “Transaction Processing in Mobile Computing Environment,” *IEEE Workshop on Advances in Parallel and Distributed Systems*, 1993, pp. 77–83.

[DER] “Apache Derby Cloudscape Overview”. Apache. [Online]. http://db.apache.org/derby/

[DUN 97] M. H. Dunham, et al, “A Mobile Transaction Model That Captures Both the Data and Movement Behavior,” *Mobile Networks and Applications*, 2(2): pp. 149–162, 1997.

[GAR 83] H. Garcia-Molina, “Using semantic knowledge for transaction processing in a distributed database,” *ACM Transactions on Database Systems*, 8(2): pp. 186–213, 1983.

[GRA 78] J. Gray, “Notes on data base operating systems,” *Operating Systems, An Advanced Course*, pp. 393–481, 1978.

[HAE 94] T. Haerder, et al., *Principles of transaction-oriented database recovery*, Morgan Kaufmann Publishers Inc., 1994.

[HAR 00] J. R. Haritsa, K. Ramamritham, “Gupta, R.: The prompt real-time commit protocol,” *IEEE Transactions on Parallel and Distributed Systems*, 11(2) pp. 160–181, 2000.

[J9] WebSphere Everyplace Custom Environment. [Online]. http://www-01.ibm.com/software/wireless/wece/

[JON 01] V. Jones, R. Bults, D. Konstantas, and P. A M Vierhout, “Healthcare PANs: Personal Area Networks for trauma care and home care,” *Fourth International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2001.

[JVT] Java ME. [Online]. http://java.sun.com/javame/

[KUM 98] V. Kumar, et al., “Defining Location Data Dependency, Transaction Mobility and Commitment,” TR 98-CSE-1, Southern Methodist University, 1998.

[KUM 00] V. Kumar, “A Timeout-Based Mobile Transaction Commitment Protocol,” *East-European Conference on Advances in Databases and Information Systems*, pp. 339–345. 2000.

[KUM 02] V. Kumar, et al., “TCOT-A Timeout-Based Mobile Transaction Commitment Protocol,” *IEEE Transactions on Computers*, 51(10): pp. 1212–1218, 2002.

[LIU 02] Y. S. Liu, G. Liao, G. Li, J. Xia, “Relaxed atomic commit for real-time transactions in mobile computing environment,” *Third International Conference on Advances in Web-Age Information Management*, pp. 397–408, 2002.

[MAD 01] S. K. Madria, et al., “A Transaction Model to Improve Data Availability in Mobile Computing,” *Distributed Parallel Databases*, 10(2): pp. 127–160, 2001.

[MOH] MobiHealth Project. [Online]. http://www.mobihealth.org/

[NOU 05] N. Nouali, et al., “A Two-Phase Commit Protocol for Mobile Wireless Environment,” *16th Australasian Database Conference*, pp. 135–143. 2005.

[OZS 91] M. T. Ozsu, et al., *Principles of distributed Database Systems*, Prentice-Hall, Inc., 1991.

[PIT 95] E. Pitoura, et al., “Maintaining consistency of data in mobile distributed environments,” *15th ICDCS*, pp. 404–413, 1995.

[SQL 92] Database Language SQL. [Online]. http://www.contrib.andrew.cmu.edu/~shadow/sql/