1998 International Computer Symposium
Workshop on Computer Networks, Internet, and Multimedia
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

Lightweight Commit Protocol for Mobile Clients

Yen-Wen Lin
Department of Computer Science and Information Engineering
Chaoyang University of Technology, Taichung, Taiwan
Feipei Lai
Department of Electrical Engineering and
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
Hsiao-Kuang Wu
Department of Computer Science and Information Engineering
National Central University, Chungli, Taiwan

Abstract

Technical advances in the development of portable computers and wireless communications enable users to take part in distributed computing even while moving. The resulting environment is subject to be constrained by the mobility of users and the nature of the cordless medium. In this paper we propose a lightweight commit protocol for providing mobile hosts with two phase commit service which is a powerful technique to implement atomic actions in distributed systems, with some important aspects such as low power consumption, efficient mobility management, subject oriented service binding and effective disconnection handling to well adapt to a mobile computing environment.

Keywords: mobile computing, wireless network, 2PC, hand-off, subject oriented service binding, disconnection

1 Introduction

A 2PC protocol is widely applied to resolve the problem of atomicity control. However, when the conventional 2PC protocol is adapted to a mobile environment, unfortunately, some issues [1, 2, 3] definitely need to be reconsidered. First, most of the conventional distributed computing paradigms take stationary hosts as a base assumption. The *mobility* of the hosts implies that they can access from any access points at any time and stay connected while on the move. The capability of supporting efficient mobility management and high quality hand-off will be essential. Second, limited battery life seriously constraints the activities of a mobile host powered by batteries. Due to this fact, *energy conservation* is an important consideration. To the greatest extent possible, we attempt to shift the workload from mobile hosts to fixed hosts. Third, due to either physical cell crossovers or power conservation, an MH may disconnect [4] itself from the rest of the system. More wisdom is needed to handle *long-duration disconnection* of mobile hosts. Forth, the characteristics of *wireless link* are often distinguished from the ones of conventional wired network. That is, the wireless link often has much lower bandwidth, higher bandwidth variability, and is monetarily more expensive. Thus, it will be rewarding to optimize the number of messages exchanged between a mobile host and its peer fixed host via wireless links.

In this paper we propose a lightweight commit protocol which supports a 2PC service for mobile clients. In this protocol, a mobile client is free to move or disconnect after submitting its commit service request to its local base station. Namely *proxy model*, the base station then takes over the responsibility to finish the request and finally delivers the result to the mobile client. Since most of workload is shifted to fixed hosts, the power consumption in a mobile host is significantly reduced. Where, through efficiently tracking the moving clients, we minimize the ill-influence resulting from the mobility of the client by seperating the ongoing service and location management functionality into two independent parts, *proxy handoff* and *service handoff*. In addition, in order to reduce message traffic and support *location-dependent* information access, the proposed *subject oriented service binding* scheme can dynamically allocate most-fit service supplier (worker) for the on-the-move client. Be-

sides, disconnection of a mobile client due to power saving or physical cell cross-over can be effectively solved by the proposed protocol.

The remainder of this paper is organized as follows. Section 2 presents the proposed commit protocol and the strategies adopted in the protocol. The correctness are described in Section 3. Some suitable applications of the proposed protocol are described in Section 4, while a brief summary is presented in Section 5.

2 Lightweight Two Phase Commit Protocol

In this section, we will introduce the system architecture, the proposed protocol and the strategies adopted in our system.

2.1 System Overview

The model, as depicted in Figure 1, is derived from [5]. A *mobile host* (MH) is a computer that can move while remaining its network connections through wireless communication. A *mobile support station* (MSS) or a *base station* is a computer augmented with a wireless interface to communicate with mobile hosts and, is connected to a fixed network through wired communication. A *cell* is a geographical coverage area serviced by an MSS. An MH can directly communicate with an MSS only if the MH is physically located within the cell serviced by the MSS via a wireless medium. The process during which a mobile host enters a new cell is called *hand-off*. All fixed hosts (including MSS) and the communication paths between them constitute the *fixed network*. Each MSS and the local MHs within the cell form a *wireless cell*. However, our system model diverges from the architecture of [5] by assuming that some of the fixed hosts are equipped with a database (as an abstraction of all needed resources maintained by a server) of specific service. A *server* is the software that runs at an MSS or a fixed host and provides mobile application services and information to the mobile hosts. The geographical coverage area for the service is called a *service area*. It is likely that a service area will cover several wireless cell.

Protocol Sketch

To adapt the conventional 2PC protocol to mobile computing environment, we extend the protocol as consequence of taking the peculiarities of a mobile host into account. As depicted in Figure 2, there are four main modules (*client, proxy, coordinator and worker*) involved in our system. The mobile host who issues the commit request to the system is called the *client*. The base station of the cell within which the mobile client currently located becomes the *current proxy* of the client. The base station who first receives the commit request becomes the *coordinator* of the commit service. The server providing needed service in the requested commit request is called the *worker*. It is possible that several workers are involved in one 2PC request. In our design, as shown in Figure 2, as the mobile client requests to commit a transaction, the local MSS that receives the request becomes the coordinator of this commit service. (Meanwhile, the MSS becomes the current proxy of this client and acts on behalf of the client.) The coordinator, then, tries to find and allocate qualified workers who physically supply the needed service. If, fortunately, all needed workers are found and promise to finish its job, the coordinator decides to really *commit* the transaction and inform all employed workers to do so. Otherwise, if the coordinator cannot find available workers to support specific service, it must inform all allocated workers to *abort* their jobs and rollback to their initial states of the transaction. After receiving ACKs from all workers, the coordinator finds the current proxy of the client and forwards the result to it. (Since the client may move to a new cell during the service session, the system needs to hand-off the state information of the mobile client from the old proxy to the new proxy.) The proxy then delivers the result to the mobile client. After getting an ACK from the mobile client, the proxy sends ACK to the coordinator, who can now release all locked resources and close the commit service session. The complete version of the proposed protocol is presented in [6].

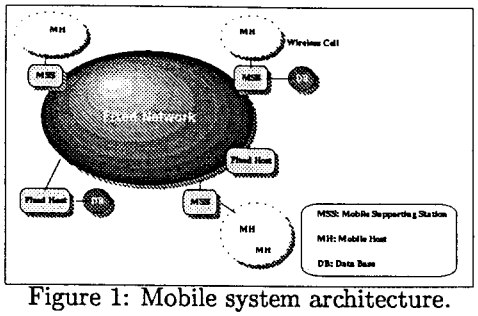

Figure 1: Mobile system architecture.

2.2 Low Power Service

Since power is such a precious resource for mobile hosts, it is totally unacceptable that a mobile client

	Client	Proxy	Coordinator	Worker I (I=1, 2, ..., n)
1	sending Mobile_Commit_Request to proxy			
2		receiving Mobile_Commit_Request from the client	the first proxy becomes the coordinator; sending Would_You to all qualified workers; waiting for replies;	
3				receiving Would_You; if can finish the job then writing undo and redo log; sending Agree; else sending Reject; waiting for response;
4			if all workers reply Agree then writing Commit log; sending Commit to all workers; else sending Abort to all workers; waiting for ACKs;	
5				if receiving Commit then releasing resource and locks; sending W_ACK; if receiving Abort then undo the request; releasing resource and locks; sending W_ACK;
6			if any W_ack is not received within a time-out period then resending Commit / Abort to the worker; if all W_ACK are received then writing Complete log; sending Commit / Abort to the proxy;	
7		receiving Commit / Abort; sending M_Commit / M_Abort to client;		
8	receiving M_Commit or M_Abort; sending M_ACK to proxy;			
9		receiving M_ACK; sends P_ACK to the coordinator;		
10			releasing locked resource; terminating the service;	

Figure 2: Sketch of the proposed lightweight 2PC protocol.

actively powers on to receive the result of the long-lived transaction processing. Due to this fact, it would be a good idea to shift most of the workload from mobile hosts to fixed hosts. In our design, the first local MSS receiving the commit request becomes the coordinator of the commit service. The coordinator coordinates the transaction processing among all the allocated workers to get a joint decision (either to commit or to abort the transaction) and keeps contact with current proxy who acts on behalf of the mobile client. That is, after sending the service requests to the local MSS, the mobile client can either go to sleep or just power off to reduce power consumption. The fixed hosts (coordinator, workers and proxy) take over all the jobs and communicate with the community via the wired network to minimize the message exchanged via expensive wireless links. Eventually, the coordinator locates the requesting mobile client and sends it the result. Thus, the workload of the client is minimized.

2.3 Efficient Mobility Management

In our design, each time the mobile client moves out of the physical cell boundary a *proxy hand-off* is needed. That is, the new proxy requests the old proxy to transfer the state of the client and also informs the coordinator the current position of the client. However, in contrast, the coordinator and all the allocated workers will not be changed correspondingly to the movement of the client before the whole commit request session completes. Hence, the ill-influence caused by frequent hand-offs on the service response time is minimized as a consequence of dividing the proxy of the mobile client and the coordinator of the service into two independent modules.

As proposed in [5], the *call hand-off* for mobile clients means the physical connection transfer between the old and new MSSes, while the *service hand-off* is used to depict the virtual connection transfer between the old and the new servers for a specific service. In our design, each time the mobile client moves out of a cell boundary a *proxy hand-off* is needed. However, when the atomicity requirement of a transaction is carefully considered, no *service hand-off* between workers is allowed after a worker is allocated by the coordinator. The crossover message is just shown as a notification to the user.

2.4 Subject Oriented Service Binding

In a subject oriented naming scheme, a client is bound to a service rather than some specific server so that the system can switch servers transparently. To facilitate the transparent naming functionality, we design a naming scheme to supply the naming service. In our system, a well-known name server is equipped with a table, *WorkerDatabase*, which collects the associated information about several kinds of service, including the service known, the jobs included in each specific service, all qualified workers for each specific job and the coverage under the control of each worker. All the information recorded in the table are collected by active registration of each worker in advance. Each worker locally keeps a *JobTable* to record the supplied jobs, the covered service area and the busy status. As mentioned above, to be allocated for a proper job, the worker should actively register itself to the specific name server in advance. The *CandidateMap* , as a subset of the WorkerDatabase, is a local cache on the coordinator as the result of querying the name server for specific service. Notice that, each sub-task included in the requested service may find several qualified workers. At the coordinator, there exists another table called *WorkerMap* which records information about the currently allocated workers who are employed by the coordinator. The information includes the service name, all the included sub-tasks, allocated worker's name, each worker's service area and the ACKed flag for indicating if it has acknowledged receiving the Commit or Abort directive.

The steps included in a service binding are depicted in Figure 3. The worker managing some specific resources needs to register itself with a well-known name server. The name server collects the associated information (into the WorkerDatabase) about several kinds of published service and the qualified workers needed for each sub-task included in these services. To employ suitable workers for requested service, the coordinator needs to query the name server to get the related information of the needed workers and records the result in the CandidateMap. Then, the coordinator invites each needed worker to do the requested service. After receiving the invitation, the requested worker evaluates (by looking up the local JobTable) if it can finish the job. Several factors (such as if the worker is busy, if the mobile client currently resides in the worker's service area and if the needed resources are available) are taken into account to evaluate the fitness. If a worker promises the coordinator, related information about the allocated worker will be recorded in the WorkerMap at the coordinator and the binding is completed. It is worth mentioning that all these efforts are transparent to the clients. All the clients need do is to submit its service request to its local MSS.

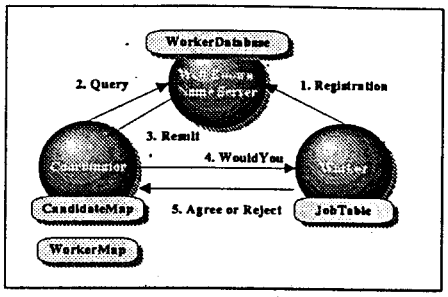

Figure 3: Service binding.

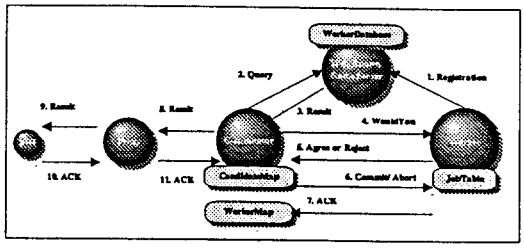

Figure 4: Flow of disconnection-free and handoff-free.

2.5 Disconnection Handling

In a mobile environment, disconnection and reconnection of a mobile host is fairly routine with cell crossovers and/or the battery power conservation [4]. Also, it could be a long time before the mobile host reconnects itself to the rest of the system. It seems to be naive just to time-out, as would be the strategy of the conventional distributed systems, if the host is unreachable in case either the host just goes down or the host powers-off to save battery power. However, we would like to obtain a result after the long-run request procedure. Fortunately, when reconsidering the 2PC protocol, you will find that it may not be necessary to keep the client involved in the process all the time. Thus it would be advantageous to develop a 2PC protocol to well fit in a mobile computing environment.

As disconnection may happen frequently, it is essential to develop a strategy to handle this situation. In our design, after the MH submitting the request, the coordinator takes over all succeeding jobs. The mobile client does not really need to be involved in the service session, thence, the mobile client can disconnect itself from the system after submission. When the transaction is finished, the coordinator forwards the result to the last known proxy that was in touched with the client. During the service session, however, the client may disconnect and later emerge in some cell (either moving to another cell or staying in the same cell) and re-establish contact with the rest of the system via its current proxy. The current proxy of the client needs to inform the coordinator and the old proxy of the current location of the client. Thus, the coordinator can send the result to the new proxy and the client. We now discuss four possible scenarios resulting from disconnection and/or hand-offs in Figure 4 to Figure 7.

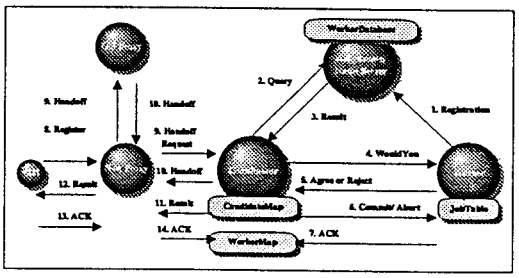

Figure 5: Flow of disconnection-free and handoff handling.

Scenario 1: Neither disconnection nor hand-off

Figure 4 shows the condition involving neither disconnection nor hand-off in the whole service progress. After receiving the result from the coordinator (step 8), the proxy simply forwards the result to the client (step 9), then waits for the ACK sent back from the mobile client (step 10) and ACKs to the coordinator (step 11).

Scenario 2: Hand-off but not disconnection

In Figure 5, no disconnection happens, but the client may move out of a cell and hand-off handling is required. That is, a mobile client moves to another cell and registers itself to the new proxy (step 8). The new proxy requests the old proxy and the coordinator to hand-off (steps 9, 10). After that, the coordinator can send the result to the new proxy (step 11). The new proxy then sends the result to the client (step 12), waits for the ACK from the client (step 13) and ACKs to the coordinator (step 14).

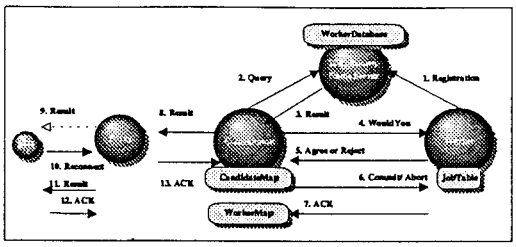

Figure 6: Flow of disconnection and handoff-free handling.

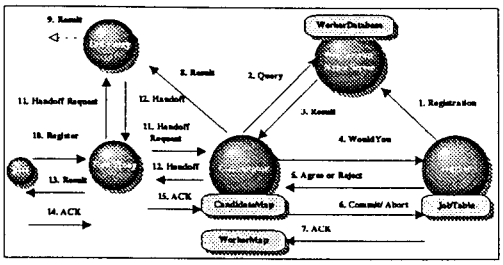

Figure 7: Flow of disconnection and handoff handling.

Scenario 3: Disconnection but not hand-off

Figure 6 illustrates the condition that the mobile client never moves out of the cell boundary, but ever disconnects itself from the proxy during the service session. In this case, the proxy tries to forward the result received from the coordinator (step 8) to the client that now breaks contact with the proxy (step 9). The proxy holds the result until the client reconnects (step 10) and forwards the result to the client (step 11). After receiving the ACK from the client (step 12), the proxy ACKs to the coordinator (step 13).

Scenario 4: Both hand-off and disconnection

Figure 7 presents the condition that the mobile client disconnects and moves out of the cell. Before the mobile client reconnects itself to the system again, the coordinator may send the result to the last known proxy last in touched with the mobile client (step 8). As the client disconnects and moves out of its cell, the proxy fails to deliver the result (step 9). Sooner or later, the mobile client comes up and registers itself to a new proxy (step 10). The new proxy then requests the old proxy and the coordinator to hand-off (step 11). Eventually the new proxy gets the result (step 12) and forwards it to the mobile client (step 13). After receiving the ACK from the client (step 14), the proxy ACKs to the coordinator (step 15).

3 Correctness

Now, we would like to claim the correctness of the protocol we proposed in this paper. Recall that, there are two aspects of atomicity of a transaction [7]. First, a transaction either completes successfully and the effects of all of its operations are saved in permanent storage or it has no effect at all. Second, the intermediate effects of a transaction must not be visible to other transactions. Where the correctness of conventional 2PC protocol has already been well proven in some existing literature [7]. Take the correctness of 2PC protocol as a base assumption, all we need to prove is to inspect whether the mobility and disconnection of the mobile client and kinds of failures during the transaction session will hurt the atomicity of a transaction service. Before going any further, it is helpful to remember that there are four modules (client, proxy, coordinator and worker) involved in our system.

3.1 Mobility

After receiving the commit request, the coordinator takes the responsibility of processing a 2PC service. During the service session, the mobile client and the proxy are not really involved in the transaction processing. Thus, they are free during the service session under one condition, that is, to keep close contact with the coordinator by reportting their current location. Thence, the coordinator can eventually forward the result to the mobile client. Though, due to the movement of the mobile client, proxy handoff may be needed during the session. The coordinator and all allocated workers are not changed by the movement of the mobile client, the result of the transaction will not be affected at all.

3.2 Disconnection and Failures

To some extent, disconnection could be considered, from the viewpoint of the rest of the system, as planned or accident failure. To focus our attention on the aspect of atomicity, either disconnection or kinds of failures would be considered as failures. And, we would like to assume that the failed components eventually would reconnect themself to the system

by default recovery procedure or by manual repair. Conventional 2PC protocol has been proven [7] that it could guarantee the atomicity of the transaction even if kinds of failures happen. As described in [6], for the sake of completeness, we discuss the correctness according to the possible failures of the main four modules included in our protocol. There are several time points that these modules may fail. We would like to claim that according to the proposed protocol, after the system recovers from failures, all the workers will achieve a coordinated decision (either to commit or to abort) and the atomicity of the transaction will be preserved. However, to save the space, we present the discussion in [6].

4 Potential Applications

As mentioned earlier, 2PC protocol is applicable to almost any multiparty operation. Now, we describe some suitable applications of our protocol.

Distributed Transaction

The data items involved in a service may be distributed among several servers and a transaction may involve multiple servers. That is a so called *distributed transaction*. Obtaining the atomicity property of transactions requires that either all of the involved servers commit the transaction or all of them abort the transaction. You can find many suitable applications of distributed transaction in the mobile environment. For example, *banking* requests issued at mobile host from somewhere may involve data items at different bank branches. Thus, a strict coordination among these branches is needed to convert the whole banking system from one consistent state to another consistent state.

Replication Management

Replication is a key to provide enhanced performance, high availability and fault tolerance in a distributed system [7, 8]. The main problem here is applying operations from clients to multiple replicas in a consistent way, while maintaining acceptable system throughput and response time. The basic requirement for replicated data is *consistency*. It is not acceptable for different clients to get different results when they access these replicated data items. Besides, *replication transparency* is another requirement when data are replicated. In other words, clients should not be aware that multiple copies of data physically exist. In our system, the coordinator takes charge of all succeeding jobs after receiving the request from the mobile client, thenceforth, replication is totally transparent to the client. Furthermore, due to the atomicity property of 2PC protocol, replica consistency can be completely satisfied.

Group Communication

A *group* is a collection of processes that act together in some way. The key property [8] that all groups have is when a message is sent to the group, all of the members of the group receive it. *Group communication* is very useful for constructing distributed systems with characteristics such as fault tolerance and/or better performance based on replicated services and multiple updates which are expected to achieve the properties of atomicity and ordering. The purpose of using a group is to allow processes to handle collections of processes as a single abstraction. Thus, a process can send a message to a group of servers without having to know where they are or how many there are. The atomicity can be perfectly achieved when our lightweight commit protocol is used.

5 Conclusion

In this paper we propose a lightweight commit protocol, which supports conventional 2PC service for mobile clients. In our system, by shifting most workload to peer fixed hosts, the load, the power consumption and the message exchanged via expensive wireless links in a mobile host are greatly reduced. Through efficiently tracking the moving clients, we minimize the ill-influence resulting from the mobility of the client by seperating the ongoing service and location management functionality into two independent parts, *proxy handoff* and *service handoff*. In addition, a subject oriented service binding scheme is designed to transparently allocate best-proper service suppliers for the mobile clients, which is essential to supply location-dependent information and significantly reduce message traffic. Disconnection can be effectively solved in the proposed protocol.

References

[1] T. Imielinski and B. R. Badrinath. Wireless mobile computing: Challenges in data management. *Communication of ACM*, 37(10):19–28, 1994.

[2] G. Forman and Zahorjan. The challenges of mobile computing. *IEEE Computer*, pages 38–47, 1994.

[3] B. R. Badrinath, A. Acharya, and T. Imirelinski. Structuring distributed algorithms for mobile hosts. In *Proc. Of the 14th Intl. Conf. On distributed Computing Systems*, pages 21–28, 1994.

[4] J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Trans. on Computer Systems*, 10(1), 1992.

[5] J. Ioannidis, D. Duchamp, and G. Q. Maguire. Ip-based protocols for mobile internetworking. In *Proc. of ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, pages 235–245, 1991.

[6] Y. W. Lin, F. P. Lai, and H. K. Wu. Lightweight commit protocol for mobile clients. Department of Computer Science and Information Engineering of National Taiwan University.

[7] G. F. Coulouris and J. Dollimore. *Distributed Systems: Concepts and Design.* Addision-Wesley, 1994.

[8] A. Goscinski. *Distributed Operating Systems: The Logical Design.* Addision-Wesley, 1991.