PAPER

Commit Protocol for Low-Powered Mobile Clients

Yen-Wen LIN[†], **Hsiao-Kuang WU**[††], ***Nonmembers, and*** **Feipei LAI**[†††], ***Member***

SUMMARY 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 commit protocol for providing low-powered 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.

key words: *mobile client, wireless network, two-phase commit protocol, mobility, hand-off, disconnection*

1. Introduction

An atomic transaction is a set of operations that is indivisible in two senses [1]. First, a transaction is all-or-nothing in its effects and the partial effects must be invisible to other transactions. Second, a transaction may involve multiple servers physically managing several kinds of resources involved in a service. Thence, extra coordination is needed to obtain the atomicity property of transactions. The most commonly used protocol known as *two-phase commit* (2PC) protocol [1], [2] allows the involved servers to reach a joint decision as to whether to commit or abort. As other distributed computing paradigms do, the conventional implementations of 2PC assume that all the hosts in a network are stationary and the power supply and the network bandwidth are freely available resources. However, the presence of mobile hosts [3], [4] invalidates all the above assumptions. In wireless environment, the clients may move at any time. The portable computers are usually equipped with limited power supply. Besides, the wireless link has much lower bandwidth, higher error rates and is much more expensive.

Manuscript received January 29, 1998.
Manuscript revised January 28, 1999.

[†]The author is with the Department of Information and Communication Engineering, Chaoyang University of Technology, Taichung, Taiwan.
[††]The author is with the Department of Computer Science and Information Engineering, National Central University, Chungli, Taiwan.
[†††]The author is with the Department of Electrical Engineering and the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

In this paper we propose a 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. 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. Besides, disconnection of a mobile client due to power saving or physical cell crossover can be effectively solved by the proposed protocol.

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

2. Related Research

Technical advances in the development of portable computers and cordless networking technology enable users to continue their computing even while moving. The concept, strategies and methodology in traditional distributed systems need to be reexamined to adapt to the new computing environment. This possibility stimulates a series of new research challenges [3]–[6].

Traditional Internet Protocol (IP) scheme is not adequate for mobile hosts, since mobile hosts are not permanently connected to the same network. Researches [7]–[9] focus on how to assign addresses to mobile hosts to route messages and on how to improve either the search cost or the update cost for locating a mobile host. As will be described in Sect. 3.3, we designed an improved tunneling scheme for managing mobility.

Disconnection [10] is very frequent in a mobile en-

vironment. Thence, file systems should support the autonomous operation of mobile hosts during disconnection. Caching [11] can be used to supply the availability when disconnection happens. Related issues such as cache coherence, pre-fetching and different caching techniques due to various available communication bandwidth should be reconsidered in mobile environments. The disconnection handling of our protocol, as will be described in Sect. 3.5, aims to effectively serve long-duration disconnected mobile clients.

The algorithms for maintaining consistency of a transaction [12] must be reconsidered under the new environment. Since disconnection happens so often, mobile hosts might download needed data prior to a disconnection to operate autonomously during disconnection. Also, various degree of consistency [13] between replicated data should be defined according to different available bandwidth. As compared to static hosts, mobile computers are much more insecure resulting from theft or physical damage. Thence, it is necessary to keep logs in the fixed network. [14] considers to build a information system for mobile clients and revise the concept of the transaction operation for the mobile environments. Dataman at Ruteger makes great efforts in replicating data onto more nearby fixed hosts for mobile clients [15] and discussing replication schemes for mobile clients [16]. As will be described in Sect. 4.1, atomicity and data consistency can be obtained using our protocol which supplying 2PC control.

A variety of research efforts have been tried in redesigning distributed algorithms for mobile clients. The paper [17] supports multicast service to mobile hosts and [18] revises the check-pointing algorithm for mobile computing. In [19], [20], the authors advise a methodology of structuring distributed algorithms for mobile clients. After considering characteristics of the new mobile/wireless environment, we revise conventional 2PC for serving mobile client.

A mobile agent is a program that can migrate from machine to machine in a heterogeneous network. The program decides when and where to migrate. It can suspend its execution at an arbitrary point, transport to another machine and resume execution on the new machine. Mobile agents have several advantages over the traditional client/server model, such as efficiency, fault tolerance, convenient paradigm and customization. There are several alternative techniques (such as queued RPC, proxy servers) that have many of these same advantages. The problem with these alternative techniques is that each one is only suitable for certain applications. A mobile-agent system on the other hand is a single, unified framework in which a wide range of distributed applications can be implemented easily and efficiently.

Where, Oracle's Mobile Agents [21] is a networking middleware that provides a basic foundation on which to build and deploy mobile applications. Oracle Mobile Agents are designed to facilitate connectivity over low bandwidth, high latency, occasionally unreliable, connections over a variety of wireless networks. Its three-tiered architectures replace session-based connection oriented computing with a low-level asynchronous, store-and-forward messaging system. In Oracle Mobile Agent, client requests for data or transactions are passed as messages to LAN-based software agents, which interact directly with servers. By de-coupling the client and the server, Oracle Mobile Agents eliminates the need for a constant connection.

Motivation

The concept of mobile agent is appealing, however, after reviewing above researches, we prefer building our system in higher layer to in communication layer for some considerations. Since most proposed solutions built in communication layer are not, for now, perfectly compatible with current network systems. Some modifications to the existing network systems are needed. Thence, these solutions are not widely accepted yet. Besides, some essential specification and standard do not achieve a commonly acceptable agreement among several different parties. Still, the infrastructure for mobile computing is not sufficiently sound enough. To supply *network-transparent* support, we revise existing solutions for distributed system problems in higher layer to adapt to current mobile environment. Moreover, in addition to efficient communication, some issues arisen in the wireless environment such as power conservation, mobility and hand-off management and disconnection handling need be re-considered for different kinds of applications. This kind of optimization could be done better in higher layer.

The 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 [3], [4], [19] 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 [10] 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.

3. Two Phase Commit Protocol for Mobile Clients

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

3.1 System Overview

The model, as depicted in Fig. 1, is derived from [7]. 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 [7] 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 Fig. 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 Fig. 3, 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

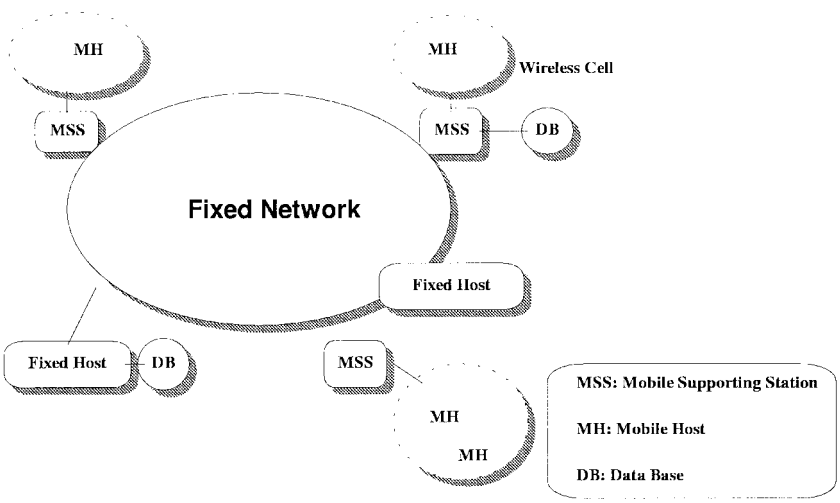


Fig. 1 Mobile system architecture.

	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;	

Fig. 2 Sketch of the proposed commit protocol.

the mobile client, the proxy sends ACK to the coordinator, who can now release all locked resources and close the commit service session.

The term *resource* used in this protocol is clossely related to what kind of application is applied to. It could be the CPU time, the memory space, needed I/O devices, data or communication bandwidth. All the four modules, especially the worker and the coordinator, need to lock resource to complete requested services. To the workers, after receiving WouldYou message from the coordinator, each invited worker needs trying to reserve and utilize resource needed to finish the requested job. If it succeed, as Step 2 in Fig. 2, it replies Agree message, otherwise Reject message. Locked resource are released after receiving indication (Commit or Abort), as Step 5 in Fig. 2, from the coordinator. To the coordinator, needed resource are locked when, as Step 2 in Fig. 2, it becomes as the coordinator of the whole 2PC service session and released when, as Step 10 in Fig. 2, it receives P-ACK from current proxy of the MH. The complete versionof the proposed protocol is presented in [22].

3.2 Low Power Service

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

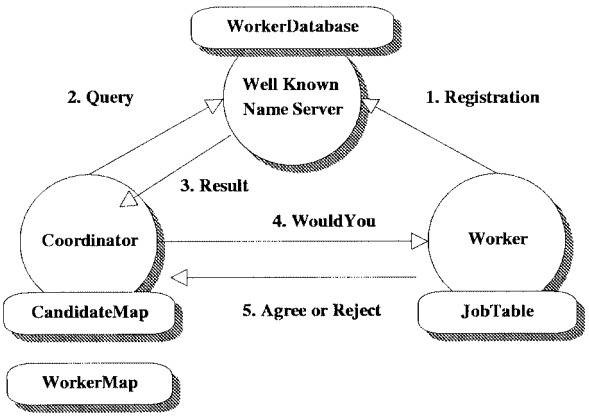


Fig. 3 Service binding.

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.

3.3 Efficient Mobility Management

In mobile computing systems, the mobile host may move anywhere at any time while retaining the connection to the network. Under the current Internet protocol, if the mobile host moves without changing its address, it will lose routing; but if it does change its address, it will lose connections. The *mobile-IP* concept [9] allows a mobile host to move without changing its IP address while retaining its connection. In one of the implementations of mobile-IP known as *tunneling* scheme [9], normal IP routing always delivers packets meant for the MH to its *home network*. When an MH is away from its home network, its *Home Agent* (HA) is responsible for intercepting and forwarding its packets. Whenever an MH moves to a *foreign network*, its *Foreign Agent* (FA) will register the current location of the MH to its HA. This scheme, in many cases, results in a non-optimal route. 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. Notice that, we implicitly embed the mobile-IP concept in our system. That is, the MH can roam freely and all of the efforts for location management are transparent to the MH. The coordinator, in our design, portrays the HA during the commit service session and the proxy acts as the FA for registering current location of MH to the coordinator. However, we do not develop our protocol based on tunneling scheme after considering the particularities of 2PC protocol. Recall that the first MSS receiving commit request becomes the coordinator of this request during the whole service session. It implies that the coordinator is employed dynamically, and it, in many cases, keeps more close to the current location of MH resulting in more efficient routing. Also, the coordinator can allocate closer-by better-fit workers even for providing location-dependent resource and/or reducing communication delay. In contrast to the location management proposed in this paper, the HA in tunneling based mobile-IP scheme is fixed and may become far away from the MH that would result in sub-optimal routing and more communication traffic. As proposed in [7], 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.

3.4 Subject Oriented Service Binding

A *server* provides a particular type of *service* to a client in a distributed system. Where a *service* is a software entity running on one or more machines, and a *server* is the service software running on a single machine. A *service binding* scheme allocates proper server for a client requesting some kind of service. To transparently supply service binding functionality in a mobile computing environment, it is essential to support *subject oriented naming* and *dynamic binding* [23]–[25]. 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.

Data Structure

In our system, a well-known name server is equipped with a table, *WorkerDatabase* (as depicted in Table 1), 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* (as illustrated in Table 2) 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 depicted in Table 3), 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* (as shown in Table 4) 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.

Service Binding

The steps included in a service binding are depicted in Fig. 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.

Table 1 Data Structure of WorkerDatabase.

Service Name	Needed Tasks	Qualified Workers	Service Area

Table 2 Data Structure of JobTable.

Job Name	Service Area	Busy ?

Table 3 Data Structure of CandidateMap.

Service Name	Needed Tasks	Qualified Workers	Service Area

Table 4 Data Structure of WorkerMap.

Service Name	Job Name	Allocated Workers	Service Area	ACK?

3.5 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 [10]. 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 Fig. 4 to Fig. 7.

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).

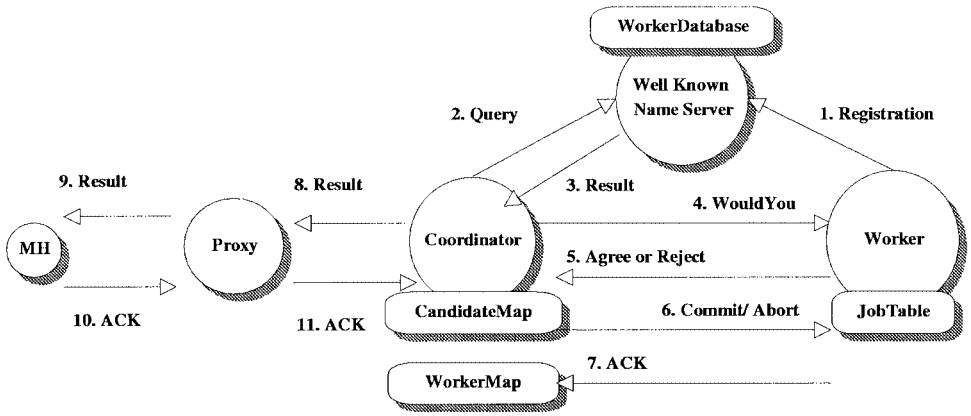


Fig. 4 Flow of disconnection-free and handoff-free handling.

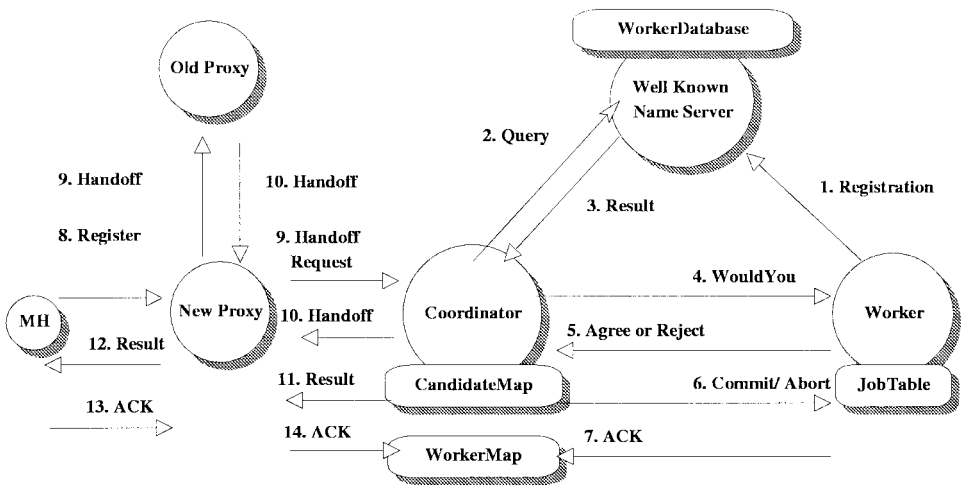


Fig. 5 Flow of disconnection-free and handoff handling.

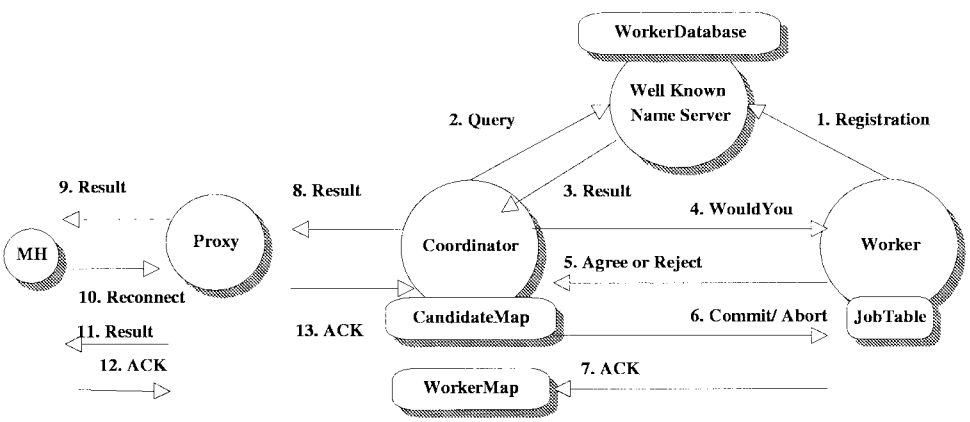


Fig. 6 Flow of disconnection and handoff-free handling.

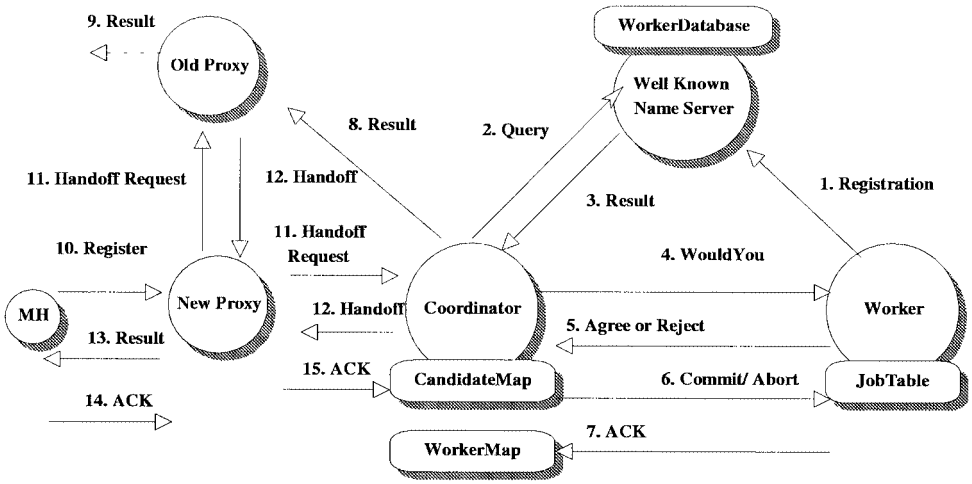


Fig. 7 Flow of disconnection and handoff handling.

Scenario 2: Hand-off but not disconnection
In Fig. 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).

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). There is one implication hidden in the protocol to be further addressed. How can the new proxy know who is respectively the coordinator and the old proxy of the mobile client that now reconnects itself to the new proxy after its disconnection from the system. In our design, the registering message IAmHere brings associated information about the previous proxy and the coordinator. So the mobile host needs to be in charge of recording enough information for future reconnection. However, if the mobile client is shocked by a sudden disconnection caused either by running out of power or by physical damage, the mobile host may fail to store sufficient critical data that is imperative to reconnection. Therefore, the mobile client may lose its memory about the previous proxy and the coordinator. To resume its commit service request, the mobile client issues a Wanted message to its new proxy. The new proxy then initiates a searching procedure to find out the previous proxy and the coordinator and requests them to do hand-off. Thenceforth, the commit service can go on. It is really a costly procedure. Fortunately, this situation does not happen so often, the searching cost could be tolerable. Certainly, under the frequent failure environment, it would be more efficient to abort the transaction if the submitting client fails during the computing when the high searching cost is considered. However, it would be worthy mentioning that the coordinator cannot easily be informed whether the client is out of contact due to sudden disconnection or it just powers off as a planned disconnection. And, more important, it is not acceptable to abort an already committed transaction when the atomicity is considered.

4. Correctness and Applications

Now, we would like to claim the correctness of the protocol we proposed in this paper and describe some possible applications of the proposed protocol.

4.1 Correctness

Recall that, there are two aspects of atomicity of a transaction [1]. 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 [1]. 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 (as introduced in Sect. 3.1 and Fig. 2).

4.1.1 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.

4.1.2 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 [1] that it could guarantee the atomicity of the transaction even if kinds of failures happen. For the sake of completeness, we briefly write down the correctness claim 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.

The Mobile Client and the Proxy Module
Since the mobile client and the proxy do not physically involve in the 2PC service, their failures during the session would not cause any real harm to the atomicity of the transaction. When, at last, they recover from failures and reconnect themselves to the rest of the system, the coordinator will find out their current location and send the result to them.

The Worker Module: Three Possible Time Points (W1, W2 and W3) to Fail

W1. The worker fails after receiving WouldYou and before it sends back Agree or Reject

- **W1.1** The coordinator times out in receiving Agree from the failed workers and sends Abort to all workers and the proxy;
- **W1.2** All workers roll back to their initial states by using *undo* log;

So, the coordinated decision among all workers is to abort the transaction. That is, the atomicity is preserved.

W2. The worker fails after it sends back Agree or Reject to the coordinator and before it receives Commit or Abort from the coordinator

- **W2.1** if all workers sends Agree
  then the coordinator sends Commit to all workers
  else the coordinator sends Abort to all workers;
- **W2.2** Since the coordinator times out in receiving WACK from the failed worker, it re-sends Commit or Abort to the failed worker;
- **W2.3** The failed worker will eventually recover from failure and roll back to the correct state by using previously stored logs.
  - **W2.3.1** if the worker receives Commit then rolls back by using *redo* log
    else /* the worker receives Abort */
    if the worker previously replied Agree to the coordinator then the worker rolls back by using *undo* log;
  - **W2.3.2** The worker sends back WACK to the coordinator;

So, either commit or abort, the final decision is consistent among all workers.

W3. The worker fails after receiving Commit or Abort and before sending back WACK to the coordinator.

- **W3.1** Since the coordinator times out in receiving WACK from the failed worker, it re-sends Commit or Abort to the failed worker;

W3.2 The failed worker will eventually recover from failure and roll back to the correct state by using previously stored logs.

W3.2.1 if the worker receives Commit then rolls back by using *redo* log else /* the worker receives Abort*/ if the worker replies Agree to the coordinator
then the worker rolls back using *undo* log;

W3.2.2 The worker sends back WACK to the coordinator;

So, either commit or abort, the final decision is consistent among all workers.

The Coordinator Module: Three Possible Time Points (C1, C2 and C3) to Fail

C1. The coordinator fails after sending WouldYou and before sending Commit or Abort to all workers.

C1.1 The workers time out in receiving Commit or Abort from the coordinator and re-sends Agree or Reject to the coordinator;

C1.2 After recovering from failure, the coordinator will evaluate the following conditions. if all workers reply Agree then sends Commit to all workers else sends Abort to all workers;

So, either commit or abort, the final decision is consistent among all workers and the atomicity is preserved.

C2. The coordinator fails after sending Commit or Abort to all workers and before receiving WACK from all workers.

C2.1 After recovering from failure, the coordinator will check the following conditions. if *commit* log can be found then rolls back to correct state by using *commit* log and re-sends Commit to all workers else re-sends Abort to all workers;

C2.2 The coordinator waits for WACK;

C2.3 if all WACK are received then sends reply (Commit or Abort) to the proxy;

So, either commit or abort, the final decision is consistent among all workers and the atomicity is preserved.

C3. The coordinator fails after receiving WACK from all workers

C3.1 After recovering from failure, the coordinator will roll back to the correct state by using *complete* log;

C3.2 if PACK is not received from the proxy then re-sends result (Commit or Abort) to the proxy;

C3.3 After receiving PACK, it releases locked resource and terminates the service session;

So, the coordinated decision among all workers (being either to abort or commit the transaction) can be replied to the proxy. Surely the atomicity is preserved.

It would be worthy of mentioning that all kinds of logs (*undo*, *redo*, *commit* and *complete* log) are stored on stable storages which could, generally be taken as reasonable assumption, guarantee the permanence of data. Surely you could replicate these logs for obtaining the capability of fault tolerance. Through the above discussion, we can conclude that the commit protocol proposed in this paper is capable of preserving sufficient atomicity as traditional 2PC protocol.

4.1.3 Timeout Handling

In this section, we would like to discuss timeout handling in the proposed protocol.

The Mobile Client Module

After sending out M-Commit-Request, the mobile client waits (in wake or sleep state) to receive M-Commit or M-Abort from current proxy. If it cannot receive reply for a long time, it can just cancel (by sending Cancel) or re-send (by sending M-Commit-Request) this commit request to the coordinator. Both messages Cancel and M-Commit-Request) are attached the same sequence number with the original commit request. Where, the sequence number is used to identify a commit request. If the coordinator receives the M-Commit-Request holding same sequence number with some previous commit request, it just neglects the request. In the meanwhile, if the result of the request is obtained, the coordinator sends back the result, otherwise, it sends back LiveWorking to the client. Both Cancel and M-Commit-Request should be sent directly to the original coordinator. Otherwise, due to the possible movement and proxy handoff, current proxy will become new coordinator and invite another set of workers. Sequence number will not work here. Thus, correctness cannot be perfectly guaranteed.

The Proxy Module

The proxy module has two possible conditions need to do timeout handling. That is, when it does not receive any reply from the coordinator module and when it does not receive any acknowledgement (M-ACK) from the mobile client. Brief discussion is as follows.

Case 1: When the proxy module does not receive reply from the coordinator for a long time:

This condition will happen due to either serious traffic delay or fails of the coordinator or some worker. In this case, certainly we can re-send the request and a series of associative complex manipulation. However, the proposed protocol, the proxy will not take any further action. Two negative side effects are considered here. As described in the last paragraph, the mobile client also

will re-send its request if it does not receive reply. Thus, more repeated request messages and more rollback messages are generated accordingly. More meaningless traffic will consume limited bandwidth. Besides, due to the movement of the mobile client, proxy handoff is possibly needed. Complicated algorithm will be introduced for time synchronization (for timing) between the new proxy and the old proxy. After second thought, in our protocol, this condition will be detected and handled by the mobile client who keeps more complete states of itself and acts better.

Case 2: When the proxy does not receive M-ACK after it forwarding the reply (M-Commit or M-Abort) to the mobile client for a long time:

This condition may happen due to the disconnection of the mobile client. In this case, the proxy will store the result in some permanent form and release locked resource and terminate this service session after sending P-ACK to the coordinator. Where, in our design, if the mobile client moves out the cell boundary during the disconnection duration, after it re-connects to the system, the old proxy will hand-over related information and the result to the new proxy.

The Coordinator Module

To the coordinator module, timeout handling may be needed for three possible conditions.

Case 1: When it does not receive reply (Agree or Reject) from some worker for a long time:

To obtain expected atomicity of the commit protocol, the coordinator just aborts all workers.

Case 2: When it does not receive W-ACK from some worker for a long time:

As described in the correctness proof earlier, the coordinator will re-send the Commit or Abort directives to the delayed worker. As a common assumption, a host (the worker module here) will recover sooner or later and receive Commit or Abort directive. Eventually, the worker will send back W-ACK to the coordinator.

Case 3: When it does not receive acknowledgement (P-ACK) from current proxy for a long time:

To save the cache capacity, after keeping the result for a reasonable long time (since the mobile client may disconnect), the coordinator will store the result in some permanent form and release locked resource and terminate the service session. Manual human operation is needed here and it is beyond our discussion.

The Worker Module

After sending Agree or Reject to the coordinator, the worker module waits for further indication from the coordinator. If the worker does not receive Commit or Abort from the coordinator for a long time, to guarantee the atomicity of the commit protocol, it will keep sending Agree or Reject to the coordinator. Again, the coordinator will recover eventually (a resonable assumption in distributed systems). Receive the message and send Commit or Abort to the worker.

Before going further, we would like to have some word about the timeout duration. As a widely accepted fact in distributed systems, the timeout duration is a complex factor and still not so clear.

To *undo* or *redo* a request, more messages will be exchanged in the system if the timeout duration is too short. On the other side, if the timeout duration is too long, related resource are locked and be idle. System resource utilization is seriously degraded. This fact is also true in a wireless mobile system. Even worse, due to the poor communication quality of wireless networks and limited battery power of a mobile host, disconnection is really routine. Long duration disconnection makes the problem more complicated. Elaborate evaluation is needed to decide proper timeout duration.

4.2 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 [1], [2]. 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 [2] 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 commit protocol is used.

5. Conclusion

In this paper we propose a 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.

Oracle's Mobile Agents is a networking middleware that provides a basic foundation on which to build and deploy mobile applications. Both our protocol and Oracle Mobile Agents are designed to facilitate connectivity over low bandwidth, high latency, occasionally unreliable, connections over a variety of wireless networks. Both architectures replace session-based connection oriented computing with an asynchronous, store-and-forward messaging system. In Oracle Mobile Agent, client requests for data or transactions are passed as messages to LAN-based software agents, which interact directly with servers. By de-coupling the client and the server, Oracle Mobile Agents eliminates the need for a constant connection. As extended three-tier model (client, agent, server), there are four modules (client, proxy, coordinator, worker) in our protocol. Combining the functionality of the proxy module and the coordinator module, our protocol can support similar concept of the mobile agent. However, to reduce the handoff overhead due to context switching, in our design, we further de-couple the agent into two independent modules (the proxy and the coordinator). In fact, there is no code migration in our protocol. The coordinator is almost fixed on some host. Only lightweight proxy handoff is needed due to the movement of the mobile client. Besides, as compared with Oracle Mobile Agent, our protocol is more specific to and focus on addressing the atomicity control among several servers (the worker module in our protocol). Because the agent (the coordinator in our protocol) has more complete information about the whole service session, it is more natural to leave the responsibility of coordination to the agent not to the server as the way of Oracle Mobile Agent. Furthermore, Oracle Mobile Agent is a low-level asynchronous store and forward messaging system. However, our protocol is developed on higher-layer to supply network-transparent support and more application-specific optimization.

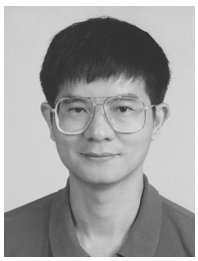
References

[1] G.F. Coulouris and J. Dollimore, Distributed Systems: Concepts and Design, Addision-Wesley, 1994.
[2] A. Goscinski, Distributed Operating Systems: The Logical Design, Addision-Wesley, 1991.
[3] T. Imielinski and B.R. Badrinath, "Wireless mobile computing: Challenges in data management," Commun. ACM, vol.37, no.10, pp.19–28, 1994.
[4] G. Forman and Zahorjan, "The challenges of mobile computing," IEEE Computer, pp.38–47, 1994.
[5] D. Duchamp, S.K. Feiner, and G.Q. Maguire, "Software technology for wireless mobile computing," IEEE Network, pp.12–18, 1991.
[6] M. Weiser, "Some computer science issues in ubiquitous computing," Commun. ACM, vol.36, no.7, pp.75–84, 1993.
[7] J. Ioannidis, D. Duchamp, and G.Q. Maguire, "Ip-based protocols for mobile internetworking," Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols, pp.235–245, 1991.
[8] B.R. Badrinath, T. Imielinski, and A. Virmani, "Locating strategies for personal communication networks," Proc. Workshop on Networking of Personal Communications Applications, 1992.
[9] A. Myles and D. Skellern, "Comparing four ip based MH protocols," Proc. 4th Jointed European Networking Conference, pp.191–196, 1993.
[10] J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," ACM Trans. Computer Systems, vol.10, no.1, 1992.
[11] C.D. Tait and D. Duchamp, "Service interface and replica management algorithm for mobile file system clients," Proc. First Intl. Conf. on Parallel and Distributed Information Systems, 1991.
[12] E. Pitoura and B. Bhargava, "Revising transaction concepts for mobile computing," Technical Report, Purdue University, 1993.
[13] E. Pitoura and B. Bhargava, "Maintaing consistency of data in mobile distributed environments," Technical Report, Purdue University, 1994.
[14] E. Pitoura and B. Bhargava, "Building information systems for mobile environments," Proc. 3rd Intl. Conf. on Information and Knowledge Management, 1994.
[15] T. Imirelinski and B.R. Badrinath, "Querying in highly mobile distributed environments," Proc. 18th Intl. Conf. on Very Large Databases, pp.41–52, 1992.
[16] B.R. Badrinath and T. Imielinski, "Replication and mobility," Proc. 2nd IEEE Workshop on Management of Replicated Data, pp.9–12, 1992.

[17] A. Acharya and B.R. Badrinath, "Delivering multicast messages in networks with mobile hosts," Proc. 13th Intl. Conf. on Distributed Computing Systems, 1993.
[18] A. Acharya and B.R. Badrinath, "Checkpointing distributed applications on mobile computers," Proc. 3rd Intl. Conf. on Parallel and Distributed Information Systems, 1994.
[19] B.R. Badrinath, A. Acharya, and T. Imirelinski, "Structuring distributed algorithms for mobile hosts," Proc. 14th Intl. Conf. on Distributed Computing Systems, pp.21–28, 1994.
[20] B.R. Badrinath, A. Acharya, and T. Imielinski, "Impact of mobility on distributed computations," Operating Systems Review, vol.27, no.2, pp.15–20, 1993.
[21] http://www.oracle.com/
[22] Y.W. Lin, F.P. Lai, and H.K. Wu, "Lightweight commit protocol for mobile clients," Technical Report, Department of Computer Science and Information Engineering of National Taiwan University.
[23] R. Chang and C.V. Ravishankar, "A service acquisition mechanism for the client/ service model in cygnus," Proc. 11th Intl. Conf. on Distributed Computing Systems, pp.90–97, 1991.
[24] D. Cheriton, "Dissemination-oriented communication systems," Technical Report, Stanford University, 1992.
[25] R. Jain and N. Krishnakumar, "Service hand-offs and virtual mobility for delivery of personal information services to mobile users," Technical Report, Bellcore, 1994.

Yen-Wen Lin is an assistant professor in Department of Information and Communication at Chaoyang University of Technology, Taiwan. She received her B.S. degree in Information and Computer Education from National Taiwan Normal University and M.S. degree in Computer Science from National Chiao Tung University respectively. She received her Ph.D. degree in Computer Science and Information Engineering from National Taiwan University and joined the faculty of Chaoyang University of Technology in 1998. Her current research interests include mobile computing, wireless personnal communication and distributed computing. She is a member of IICM and IEEE.

Hsiao-Kuang Wu is an assistant professor of Computer Science and Information Engineering at National Central University, Taiwan. He received his B.S. degree in Computer Science and Information Engineering Department from National Taiwan University in 1989. He received his Master and Ph.D. in Computer Science from University of California, Los Angeles (UCLA) in 1993 and 1997. His primary research interests include Wireless Network, Mobile Computing, Broadband Network.



Feipei Lai received a B.S.E.E. degree from National Taiwan University in 1980, and M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1984 and 1987, respectively. He is a professor in the Department of Electrical Engineering and in the Department of Computer Science and Information Engineering at National Taiwan University. He was also a visiting senior computer system engineer in the Center for Supercomputing Research and Development at the University of Illinois at Urbana-Champaign. Dr. Lai holds four Taiwan patents and two USA patents currently. He served as a consultant at ERSO, ITRI during 1988 and at Faraday Technology Corp. from 8/94 to 7/95. His current research interests are high performance microprocessor chip design, computer architecture, optimizing compiler, VLSI design, and artificial neural network applications. Prof. Lai is one of the founders of the Institute of Information and Computing Machinery. He is also a member of Phi Kappa Phi, Phi Tau Phi, ACM, The Chinese Institute of Engineers, The Chinese Institute of Electrical Engineers, and The Institute of Electronics, Information and Communication Engineering. He received Acer awards five times in 1989, 1991, 1992, 1993 and 1995 and The Taiwan Fuji Xerox Research award in 1991. Dr. Lai is a Senior member of IEEE and included in "Who's Who in Science and Engineering" and "Who's Who in the World."