1

Mobile Database System

1.1 INTRODUCTION

The objective of this chapter is to highlight the current trends in information management discipline. It begins with a brief history of mobile and wireless communication technology and reviews a number of milestones.

Mobility – The Most Desirable Environment

Information retrieval by users with mobile devices such as cell phones, PDA (Personal Digital Assistant), MP3 music players, etc., has become a common everyday activity. Navigational systems in vehicles are now a standard accessary like music system. These gadgets are quite useful and user-friendly because they can retrieve desired information from databases from anywhere through wireless channels. However, they have a serious limitation: The information flow in these systems is only from the server to users. This limitation does not allow users to query or manipulate the database which can be located anywhere in the world. Consequently, users just have to contend with what the server sends them, which may not always be accurate or up to date. In database terminology these system are not capable of managing transactional activities.

Database researchers, practitioners, and commercial organizations have a common vision of building an information management system on a mobile platform which is capable of providing full transaction management and database functionality from

anywhere and anytime. The recent advances in mobile discipline clearly indicate that reaching this ambitious goal is around the corner.

Traditionally, database is processed by immobile processing units: servers or clients. The spatial coordinates of these processing units are fixed, and users go to them with their data processing requests. Under this information management model, both the processing units and their users are immobile at the time of data processing. This way of managing information has some inherent efficiency problems leading to unacceptably low productivity, and it is not scalable because it is unable to grow with the present-day information processing needs. Recent changing social structure, necessity to have stronger connectivity among national and international communities, increasing spatial mobility, and fear of isolation have generated a very different class of information processing needs and demands. One of the important aspects of these demands is that a user must be free from temporal and spatial constraints in processing the desired information which can only be achieved by geographical mobility during data processing. The inherent immobility of processing units of legacy systems was a serious impediment in achieving the desired objective. A restricted type of mobility is possible to achieve in conventional systems (TV, music systems, etc). For example, a remote control unit can be connected to the system with a longer cable to turn the system on and off from anywhere in the room. Such arrangement did work and was used in many audio systems. However, such cable-assisted mobility was quite troublesome rather than a convenience and the kind of mobility which is free from visible connecting cables was urgently needed.

The introduction of mobility actually happened through remote control units. A glance at the history of remote controllers reveals interesting facts [1, 2]. The first remote control unit to activate remote machines was used in Germany, where the German navy used it to ram enemy ships in World War I. In World War II remote control units were used to detonate bombs and and activate weapons. In the United States, at the end of the wars, engineers experimented with household activities and in the late 1940's introduced automatic garage door openers, which actually marks the beginning of wireless era in the United States.

In 1952 Zenith developed a remote control called *Lazy Bones*, but it was not a mobile device. It was rather connected to the TV set with a long cable. In 1955 a unit called *Flash-o-Matic* was introduced, which activated units by throwing light on light-sensitive cells connected to TV sets. In 1957 Zenith introduced a wireless remote controller called *Space Command*, which used ultrasonic as an activation medium. The partial success achieved through ultrasonic motivated the use of infrared to activate TV sets through remote control unit, which is now an integral part of a large number of consumer electronics products such as VCRs, stereo systems, electronic toys, and computer keyboards, to name a few.

On the communication arena, the history of mobility is equally interesting. The first mobile radio, capable of one-way communication, was developed by Detroit Police in 1928 [3]. Police passenger cars, referred to as *cruisers*, were equipped with radio receivers, which were used to carry detectives and patrol officers. In 1933 two-

way radio communications were introduced, which was first used by the Bayonne, N.J. police department.

The use of mobile radio systems spread fast, and it became necessary to control the use of radio frequencies. In 1934 the United States Congress created the Federal Communications Commissions (FCC), which, in addition to regulating land-line telephone systems, also managed the use of these frequencies. In 1935 Frequency modulation was invented and developed by a Columbia University professor Maj. Edwin H. Armstrong, which was used to improve the mobile radio communication. In 1940 new frequencies between 30 and 40 MHz were made available by the FCC, which provided the necessary resources to companies and individuals to operate their own mobile units. In the same year the Connecticut State Police at Hartford and the majority of police systems around the country converted to FM technology. This marked the birth of mobile telephony.

On June 17, 1946 in St. Louis, AT&T together with Southwestern Bell made available the first commercial mobile radio-telephone service to private customers where mobile users were connected to a public switched telephone network (PSTN). Their system operated on six channels in the 150-MHz band with a 60-kHz channel spacing, but undesirable channel interference (e.g., cross-talk in a land-line phone) soon forced Bell to use only three channels.

Cellular concept originated at Bell Laboratories in 1947 and AT&T began operating a radio telephone that provided service referred to as *Highway service* between New York and Boston. This service operated in the 35- to 44-MHz band. This was a very basic mobile service where a subscriber was given one specific channel for communication. In the same year, the Bell company requested FCC for more frequencies, which was granted in 1949. However, the FCC distributed these frequencies among a number of companies, thus creating a competition among them for improving the quality of service. This helped to increase the number of mobile units significantly and set the need for automatic dialing capability.

The first fully automatic radiotelephone service started in Richmond, Indiana, on March 1, 1948, which eliminated human operator intervention for placing calls. In the same year (July 1, 1948) the Bell System introduced transistors (a joint invention of Bell Laboratories scientists William Shockley, John Bardeen, and Walter Braqttain), which revolutionized every aspect of telephone and communication industries.

By 1950s the Paging systems began to appear and the first phone-equipped car glided on the road in Stockholm, Sweden – the home of Ericsson's corporate headquarters. The first user of this system was a doctor-on-call and a bank-on-wheels. Tom Farley [3] narrates "The apparatus consisted of receiver, transmitter and logic unit mounted in the trunk of the car, with the dial and handset fixed to a board hanging over the back of the front seat. It was like driving around with a complete telephone station in the car. With all the functions of an ordinary telephone, the telephone was powered by the car battery. Rumor has it that the equipment devoured so much power that you were only able to make two calls - the second one to ask the garage to send a breakdown truck to tow away you, your car, and your flat battery. These first car phones were just too heavy and cumbersome - and too expensive to use for more than a handful of subscribers. It was not until

the mid-1960's that new equipment using transistors were brought into the market. Weighing a lot less and drawing not so much power, mobile phones now left plenty of room in the trunk—but you still needed a car to be able to move them around."

In 1956 the Bell System began offering manual radio-telephone service at 450 MHz, a new frequency band assigned to relieve overcrowding. In 1958 the Richmond Radiotelephone Company improved their automatic dialing system by adding new features to it, which included direct mobile to mobile communications. Other independent telephone companies and the Radio Common Carriers made similar advances to mobile-telephony throughout the 1950s and 1960s.

In 1964 the Bell System introduced Improved Mobile Telephone Service (IMTS), which consisted of a broadcast system equipped with a higher-power transmitter. IMTS succeeded by the badly aging Mobile Telephone System. It worked in full duplex so people didn't have to press a button to talk. Talk went back and forth just like a regular telephone. It finally permitted direct dialing, automatic channel selection, and reduced bandwidth to 25-30 kHz.

In 1970 the Federal Communication Commission (FCC) allocated spectrum space for cellular systems and by 1977 AT&T and Bell Laboratories together developed and began testing of a prototype cellular system. In 1978 public trials of the new system were started in Chicago with over 2000 trial customers, and in 1979 the first commercial cellular telephone system became operational in Tokyo. In 1981, Motorola and American Radio telephone started a second U.S. cellular radio-telephone system test in the Washington/Baltimore area. By 1982, the slow-moving FCC finally authorized commercial cellular service for the USA. A year later, the first American commercial analog cellular service or AMPS (Advanced Mobile Phone Service) was made available in Chicago for public use [4].

In 1985 Total Access Communication System (TACS) was introduced in the United Kingdom. It is the European version of AMPS and occupies the 900-MHz frequency band with an RF channel spacing of 25-kHz. ETACS was an extended version of TACS with more channels. TACS and ETACS are now obsolete and are replaced by the more scalable and all-digital Global System for Mobile communications (GSM). TACS was the first real vehicle-mounted mobile communications system, but later developed into mobile units. In the same year, CNETZ was introduced in Germany and Radiocom 2000 was deployed in France.

Until now, all system were based on analog communication, which had a number of limitations. To eliminate some of these limitations in 1987 to 1995, new air interface protocols such as TDMA (Time–Division Multiple Access), CDMA (Code–Division Multiple Access), etc., were introduced. Today's mobile systems are mainly based on digital technology, but analog systems are in use too. The Table 1.1 chronology lists important events in mobile communication.

The mobile phones and communication managed to establish a *partially connected information space*, which was free from spatial and temporal constraints. Thus, the "anytime and any place" connectivity paradigm for voice became very common.

Table 1.1 Important events in mobile communication

Date	Event
1867	Maxwell speculated the existence of electromagnetic waves.
1887	Hertz showed the existence of electromagnetic waves.
1890	Branly developed technique for detecting radio waves.
1896	Marconi demonstrated wireless telegraph.
1897	Marconi patented wireless telegraph.
1898	Marconi awarded patent for tuned communication.
1898	Wireless telegraphic connection between England and France established.
1901	Marconi successfully transmits radio signal from Cornwall to Newfoundland.
1909	Marconi received Nobel prize in physics for Voice over Radio system.
1928	Detroit police installed mobile receivers police patrol cars.
1930	Mobile transmitters were deployed in most cars.
1935	Armstrong demonstrated Frequency modulation (FM) scheme.
1940	Majority of police systems converted to FM.
1946	Mobile systems were connected to Public Switched Telephone Network (PSTN).
1949	FCC recognizes mobile radio as new class of service.
1950	Number of mobile users increased more than 500,000.
1960	Number of mobile users grew more than 1.4 million.
1960	Improved Mobile Telephone Service (IMTS) introduced.
1976	Bell Mobile used 12 channels to support 543 customers in New York.
1979	NTT/Japan deploys first cellular communication system.
1983	Advanced Mobile Phone System (AMPS) deployed in the United States.
1989	GSM appeared as European digital cellular standard.
1991	US Digital Cellular phone system introduced.
1993	IS-95 code-division multiple-access (CDMA) digital cellular system deployed in the United States.
1994	GSM Global System for Mobile Communications deployed in the United States.
1995	FCC auctioned band 1.8-GHz frequencies for Personal Communications System (PCS).
1997	Number of cellular telephone users in the United States increased to 50 million.
2000	Third-generation cellular system standards? Bluetooth standards?

1.1.1 Fully Connected Information Space

Figure 1.1 introduces the concept of a *fully connected information space* that people envision today. Each object of this real world with some functionality is connected to other object through wireless link. For example, a bank or a person is connected to conference, bus, submarine, shark, and so on, with bidirectional wireless link. Thus at any moment a person or a bank or a scuba diver can have complete information about all other objects. Such wireless link has become essential for this highly mobile and dynamic society. Consider the case of a working parents. Their children are in different schools, and each parent works at a different place. Each member of the family would like to have instant access to the situation of their children to reach them at the time of need. Similarly, the president of a company would like to have complete information about all activities of his company to manage it efficiently.

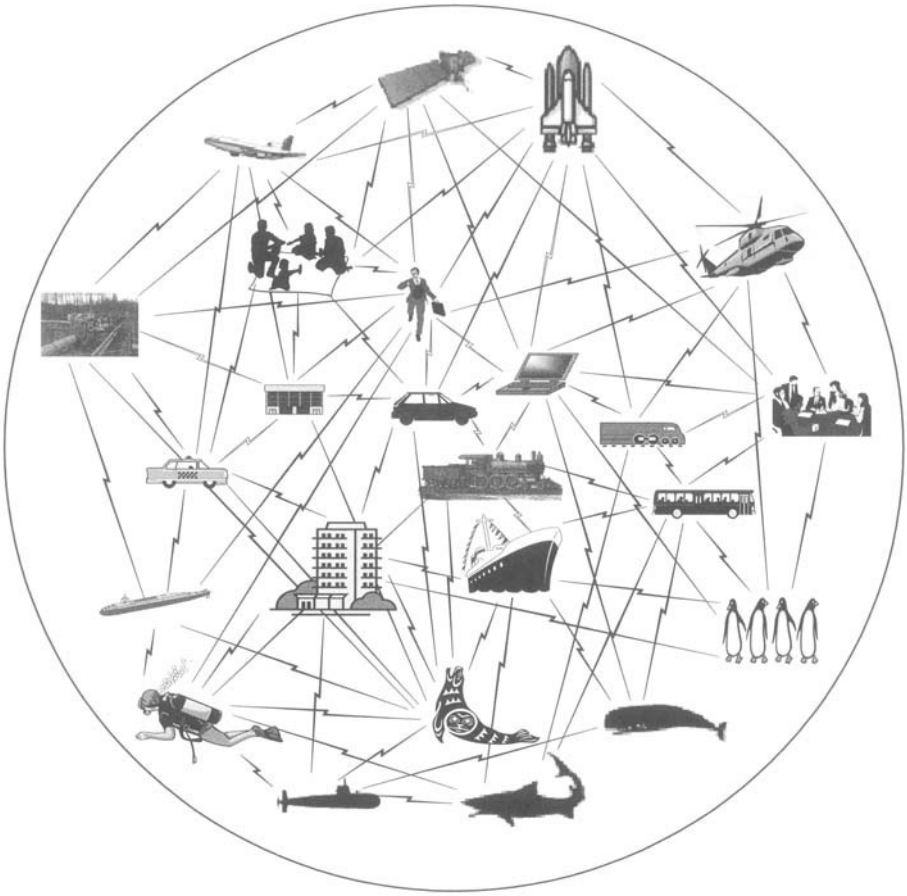

Fig. 1.1 A fully connected information space.

However, this paradigm could not allow "anytime and any place" data processing capability, which is an outstanding demand from users and industries alike. Users desire that a mobile unit (cell phone, PDA, etc.) should have transaction management capability, which will allow a user to perform everyday activities such as fund transfer, seat reservation, stock trading, etc., and in addition to this they should be able to access any information form anywhere in any state: mobile or static. Thus, a user should be able to access his or her account information, be able to pay bills, be able to buy and sell shares, etc., and allow a CEO to access his company's database and offer salary raises to its employees while traveling on a car or on a plane.

These demands and creative thinking laid down the foundation of "Ubiquitous Information Management System" or "Mobile Database System (MDS)" which in essence is a distributed client/server database system where the entire *processing environment* is mobile. The actual database may be static and stored at multiple sites but the data processing nodes, such as laptop, PDA, cell phones, etc., may be mobile and can access desired data to process transactions from anywhere and at any time.

The fully connected information space, in addition to wireless communication, needs transactional services. Today each individual likes to have facility to manage information related to him. For example, a user would like to change his personal profile for adding new call option on his cell phone service. The user would prefer to have editing capability to edit his profile to incorporate new option himself instead of reaching to the service provider. A customer would prefer to have facility to execute a fund transfer transaction himself from anywhere to pay for his purchases or to transfer money among his multiple accounts instead of requesting his bank to do so.

The mobile discipline defines two types of mobility: (a) terminal mobility and (b) personal mobility. Each mobility type addresses a different set of mobility problems.

1.2 TYPES OF MOBILITY

A mobile framework is composed of wired and wireless components and human users. Its wireless part implements *terminal mobility* and *personal mobility* to eliminate some of the spatial and temporal constraints from data processing activities.

Terminal Mobility: It allows a mobile unit (laptop, cell phone, PDA, etc.) to access desired services from any location while in motion or stationary, irrespective of who is carrying the unit. For example a cell phone can be used by its owner and it can also be borrowed by some one else for use. In terminal mobility, it is the responsibility of the wireless network to identify the communication device. Figure 1.2 illustrates the notion of terminal mobility. A person at location C (longitude/latitude = C) uses the mobile unit to communicate with the car driver at location A. He can still establish communication with the driver from a new location D irrespective of the movement of the car from A to B. The use of a phone card works on this principle. It can be used from different locations and from different machines such as pay phones, residential phones, etc.

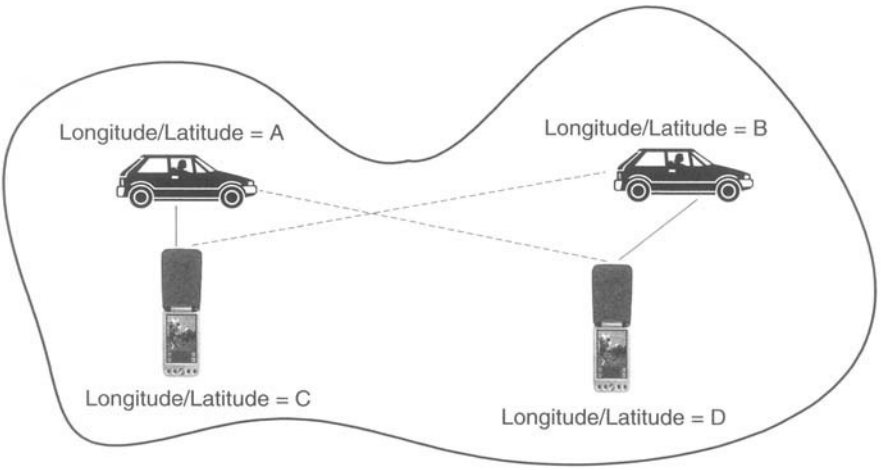

Fig. 1.2 Terminal mobility.

In terminal mobility, from a telecommunication viewpoint, the network connection point (referred to as a network access/termination point) is identified as not the called party. Thus, the connection is established between two points and not between the two persons calling each other. This type of connection in a session allows the use of communication devices to be shared among anybody.

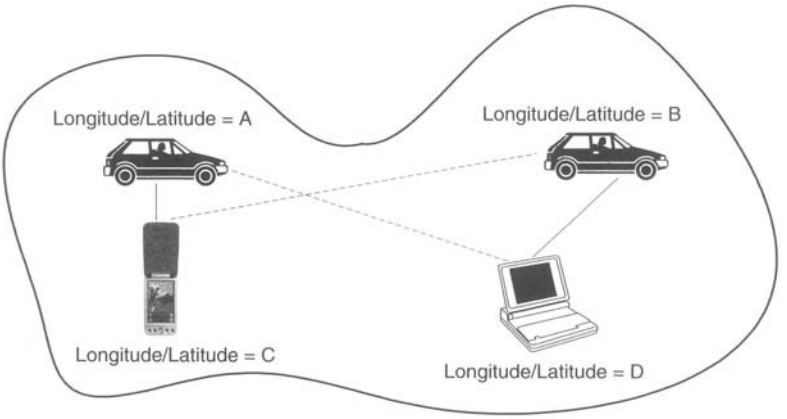

Fig. 1.3 Personal mobility.

Personal Mobility: In terminal mobility the mobility of a terminal is supported; that is, the same terminal can be used to connect to the other party from anywhere by any user. In personal mobility this capability is provided to a human being. Thus, a user does not have to carry any communication equipment with him; he

can use any communication device for establishing communication with the other party. This facility requires an identification scheme to verify the person wishing to communicate. Figure 1.3 illustrates the notion of personal mobility. A person at location C communicates with the car at location A using his PDA, and from location D also he can communicate with the car at location A using his laptop. At present, personal mobility is available through the web. A user can log on to the web from different machines located at different places and access his e-mail. The mobile system extends this facility so that the user can use any mobile device for reaching the internet. In personal mobility each person has to be uniquely identified, and one way to do this is via a unique identification number.

There is no dependency relationship between terminal and personal mobility; each can exist without the other. In personal mobility the party is free to move, and in terminal mobility the communication unit is free to move.

Voice or data communication can be supported by either types of mobility. However, to visualize a complete mobile database management system both types of mobility are essential.

1.3 SUMMARY

This chapter covered historical facts and the emergence of mobile and wireless disciplines and wireless gadgets starting from remote control units. It discussed the types of mobility necessary to visualize mobile infrastructure and envisioned the development of a fully connected information space where all functional units are fully connected with each other through wireless links. It presented the rationale for the development of a mobile database system necessary to manage all information management tasks in the information space.

The entire development can be looked at in terms of analog and digital transmission and data transmission aspects also. The first-generation wireless technology which was basically analog is usually referred to as *First Generation (1G)*. 1G systems were deployed only in the business world in the 1980's. Mobile and cordless phones were introduced and analog standards were defined. A number of wireless communication companies such as Nokia (Finland), Motorola (USA), and Ericsson (Sweden), to name a few, established their firm hold in the communication market.

The popularity of analog wireless technology motivated users to present new demands on the system and soon the limitations of 1G infrastructure became known. In early 1990's, therefore, the second generation (2G) wireless technology was introduced which was based on digital transmission. Digital technology provided higher communication capacity and better accessibility. This marked the introduction of Global System for Mobile Communication – *Groupe Special Mobile* (GSM). Initially GSM was confined to Europe gradually its standard spread to most other countries of the world. The 2G mobile units could send not only voice but limited amount of data as well.

The limited amount of data communication capability became one of its serious limitations of 2G system. A number of more powerful mobile phones were introduced

in early 2000's, which allowed higher voice and data transmission rates and improved connectivity. This was only a partial enhancement to 2G systems, so it was referred to as "2.5G" technologies. This allowed e-mails to be received and sent through 2.5G mobile phones which could be connected to laptop or PDA (Personal Digital Assistant).

2.5G technology and system was not quite capable of handling multimedia data transfer, unrestricted internet access, video streaming, etc. These kind of transfers became very important for M-commerce community. The Third-Generation (3G) technology made it possible to achieve these capabilities. 3G made it possible to provide variety of services through internet and the emphasis moved from voice-centric to data-centric environment. It also helped to establish a seamless integration of business and user domains for the benefit of the entire society. Thus, 3G technology made it possible to visualize *fully connected information space*.

The next chapter further discuses mobility and wireless communication technology necessary to build the desired mobile database system.

Exercises

1. What is the difference between wireless communication and mobile communication? Explain your answer and give some real-life example to illustrate the differences.

2. Explain the differences between personal mobility and terminal mobility. How do they affect the scope of wireless communication?

REFERENCES

1. R. C. Goertz, "Fundamentals of General-Purpose Remote Manipulators," *Nucleonics*, Vol. 10, No. 11, Nov. 1952, pp. 36–45.

2. R. C. Goertz, "Electronically Controlled Manipulator," *Nucleonics*, Vol. 12, No. 11, Nov. 1954, pp. 46–47.

3. http://www.privateline.com/PCS/hisory5.htm.

4. http://inventors.about.com/library/weekly/aa070899.htm.

3

Location and Handoff Management

3.1 INTRODUCTION

The handoff process in mobile communication system was briefly introduced in Chapter 2. In this chapter, further details of the handoff process is provided and the topic of location management is introduced. It first explains how these processes work and then discusses their relevance to transaction management in mobile database systems. Quite a few location management schemes have been proposed recently, but none of them have been implemented in any commercial system, so they are not discussed. The working of existing handoff and location mechanisms given in IS-41 is explained [7].

3.1.1 Location Management

In cellular systems a mobile unit is free to move around within the entire area of coverage. Its movement is random and therefore its geographical location is unpredictable. This situation makes it necessary to locate the mobile unit and record its location to HLR and VLR when a call has to be delivered to it. Thus, the entire process of the mobility management component of the cellular system is responsible for two tasks: (a) location management– that is, identification of the current geographical location or current point of attachment of a mobile unit which is required by the MSC (Mobile Switching Center) to route the call– and (b) handoff– that is, transferring (handing off) the current (active) communication session to the next base station, which seamlessly resumes the session using its own set of channels. The entire process of location

management is a kind of directory management problem where locations are current locations are maintained continuously.

One of the main objectives of efficient location management schemes is to minimize the communication overhead due to database updates (mainly HLR) [6, 9, 15]. The other related issue is the distribution of HLR to shorten the access path, which is similar to data distribution problem in distributed database systems. Motivated by these issues, recently a number of innovative location management schemes have appeared in the research world [14].

The current point of attachment or location of a subscriber (mobile unit) is expressed in terms of the cell or the base station to which it is presently connected. The mobile units (called and calling subscribers) can continue to talk and move around in their respective cells; but as soon as both or any one of the units moves to a different cell, the location management procedure is invoked to identify the new location.

The unrestricted mobility of mobile units presents a complex dynamic environment, and the location management component must be able to identify the correct location of a unit without any noticeable delay. The location management performs three fundamental tasks: (a) location update, (b) location lookup, and (c) paging. In location update, which is initiated by the mobile unit, the current location of the unit is recorded in HLR and VLR databases. Location lookup is basically a database search to obtain the current location of the mobile unit and through paging the system informs the caller the location of the called unit in terms of its current base station. These two tasks are initiated by the MSC.

The cost of update and paging increases as cell size decreases, which becomes quite significant for finer granularity cells such as micro- or picocell clusters. The presence of frequent cell crossing, which is a common scenario in highly commuting zones, further adds to the cost. The system creates *location areas* and *paging areas* to minimize the cost. A number of neighboring cells are grouped together to form a location area, and the paging area is constructed in a similar way. In some situations, remote cells may be included in these areas. It is useful to keep the same set of cells for creating location and paging areas, and in most commercial systems they are usually identical. This arrangement reduces location update frequency because location updates are not necessary when a mobile unit moves in the cells of a location area. A large number of schemes to achieve low cost and infrequent update have been proposed, and new schemes continue to emerge as cellular technology advances.

A mobile unit can freely move around in (a) *active mode*, (b) *doze mode*, or (c) *power down mode*. In active mode the mobile actively communicates with other subscriber, and it may continue to move within the cell or may encounter a handoff which may interrupt the communication. It is the task of the location manager to find the new location and resume the communication. In doze mode a mobile unit does not actively communicate with other subscribers but continues to listen to the base station and monitors the signal levels around it, and in power down mode the unit is not functional at all. When it moves to a different cell in doze or power down modes, then it is neither possible nor necessary for the location manager to find the location.

The location management module uses a two-tier scheme for location-related tasks. The first tier provides a quick location lookup, and the second tier search is initiated only when the first tier search fails.

Location Lookup

A location lookup finds the location of the called party to establish the communication session. It involves searching VLR and possibly HLR. Figure 3.1 illustrates the entire lookup process [8], which is described in the following steps.

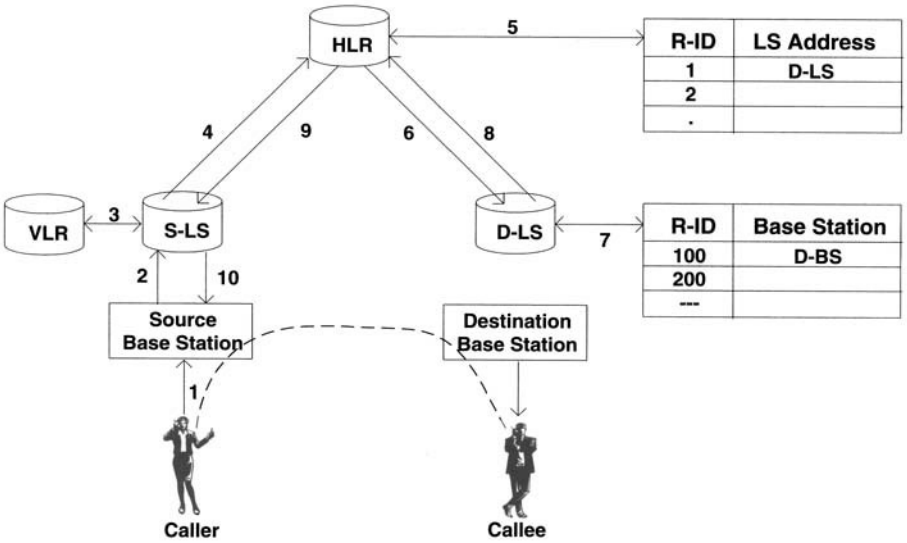

Fig. 3.1 Location search steps.

Step 1: The caller dials a number. To find the location of the called number (destination), the caller unit sends a location query to its base station **source base station**.

Step 2: The source base station sends the query to the S-LS (source location server) for location discovery.

Step 3: S-LS first looks up the VLR to find the location. If the called number is a visitor to the source base station, then the location is known and the connection is set up.

Step 4: If VLR search fails, then the location query is sent to the HLR.

Step 5: HLR finds the location of D-LS (destination location server).

Step 6: The search goes to D-LS.

Step 7: D-LS finds the address of D-BS (destination base station).

Step 8: Address of D-BS is sent to he HLR.

Step 9: HLR sends the address of D-BS to S-LS (source location server).

Step 10: The address of D-BS is sent to the source base station, which sets up the communication session.

Location Update

The location update is performed when a mobile unit enters a new registration area. A location update is relatively expensive, especially if the HLR is distributed. The frequency of updates depends on the intercell movement pattern of the mobile unit such as highly commuting subscribers. One of the tasks of a good location management scheme is to keep such updates to a minimum.

In the new registration area the mobile unit first registers with the base station, and the process of location update begins. Figure 3.2 illustrates the basic steps of location update.

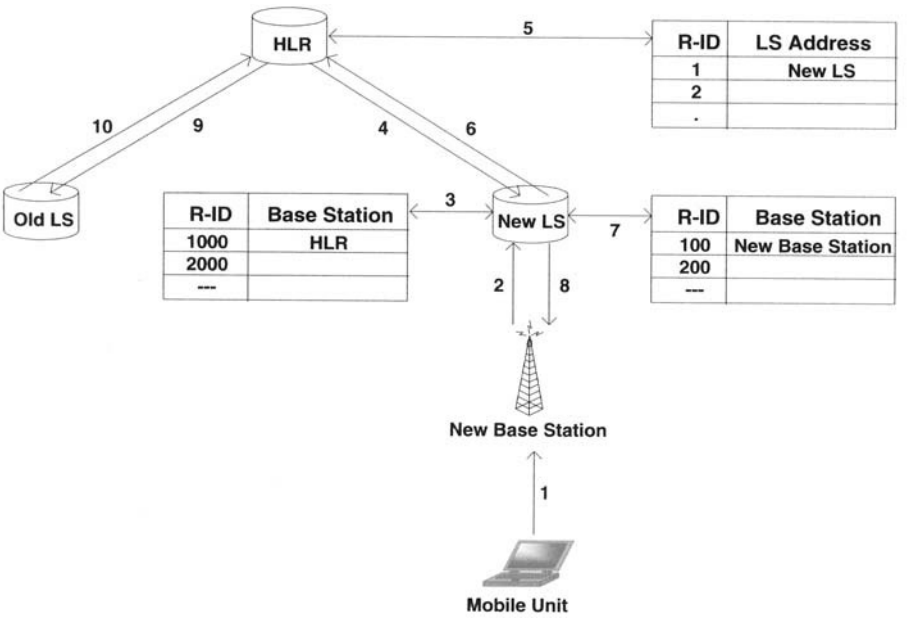

Fig. 3.2 Location update steps.

Step 1: The mobile unit moves to a new registration area which is serviced by a new location server (New LS). The mobile unit informs the new base station about its arrival.[1]

Step 2: The new base station sends the update query to New LS.

Step 3: The New LS searches the address of the HLR in its local database.

Step 4: The new location of the mobile unit is sent to HLR.

Step 5: The old location of the mobile unit is replaced by the new location.

Step 6: The HLR sends user profile and other information to New LS.

Step 7: The New LS stores the information it received from HLR.

Step 8: The New LS informs the new base station that location update has been completed.

Step 9: The HLR also sends a message about this location update to the Old LS. The Old LS deletes the old location information of the mobile unit stored in its database.

Step 10: The Old LS sends a confirmation message to the HLR.

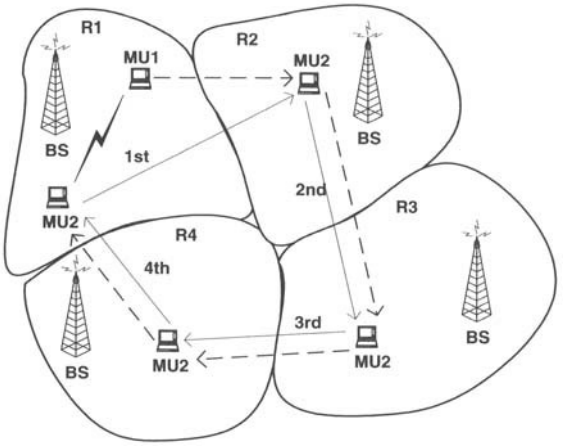

Fig. 3.3 Transient loop in forward pointer scheme.

The current location management scheme has very high search and update costs, which increase significantly in the presence of frequent cell crossing because every

[1]This is a part of registration process.

registration area crossing updates HLR. These issues motivated researchers to find efficient and cost effective schemes. A number of new location management schemes have been proposed recently, and a partial list is given here [1, 2, 3, 4, 5, 10, 11, 13]. A good survey of some these schemes can also be found in [8, 12]. Instead of presenting a particular scheme a general description of forwarding pointer approach is discussed here to present the main idea [5, 8].

Forwarding Pointer Location Management Scheme

The objective of the forwarding pointer scheme is to minimize network overhead due to HLR updates. Unlike conventional scheme, this scheme uses a pointer to the next location of the mobile user. Thus instead of updating HLR, the scheme just sets a pointer at the previous location of the mobile unit which points to its current location. The pointer is a descriptor which stores mobile unit identity and its current location. A mobile unit movement is unpredictable, and it is possible that the unit may visit a registration area multiple times during a live communication session. If forward pointers are continuously created and maintained, then a revisit to a registration area creates a transient loop. Figure 3.3 illustrates the formation of transient loop in forward pointer strategy. Initially, mobile units MU1 and MU2 were communicating in registration area **R1**. Unit MU2 makes its first move to R2, and then it moves back to R1 through R3 and R4. This type of movement creates a transient loop where the communication path is R1 $\rightarrow$ R2 $\rightarrow$ R3 $\rightarrow$ R4 $\rightarrow$ R1. However, even in the worst-case scenario the transient loop does last for long.

Updates Using Forward Pointers: When MU2 leaves registration area R1 and moves to R2 then (a) the user profile (MU2 profile) and the number of forward pointers created so far by MU2 is transferred from R1 to R2 and (b) a forward pointer is created at R1 which points to R2. This forward pointer can be stored in any BS data structure.

At some point the current location of the MU needs to be updated in HLR. Usually, heuristic based update approach is used. One scheme could be based on the number of pointers created [8]. In this scheme an upper limit of pointers can be predefined; and once this threshold is reached HLR is updated. Another scheme can be based on the number of search requests, yet another can be based on constant update time. Thus the HLR is updated after so many hours or minutes have elapsed since the last update. The performance of these update schemes will very much depend on the user mobility.

Location Search Using Forward Pointers: The search scheme is illustrated in Figure 3.4. A user in "Source" registration area wants to communicate with a user in "Destination" area. The following steps describes the location discovery.

Step 1: The caller dials the number of destination user. To find the location of the called number (destination), the caller unit sends a location query to its base station **source base station**.

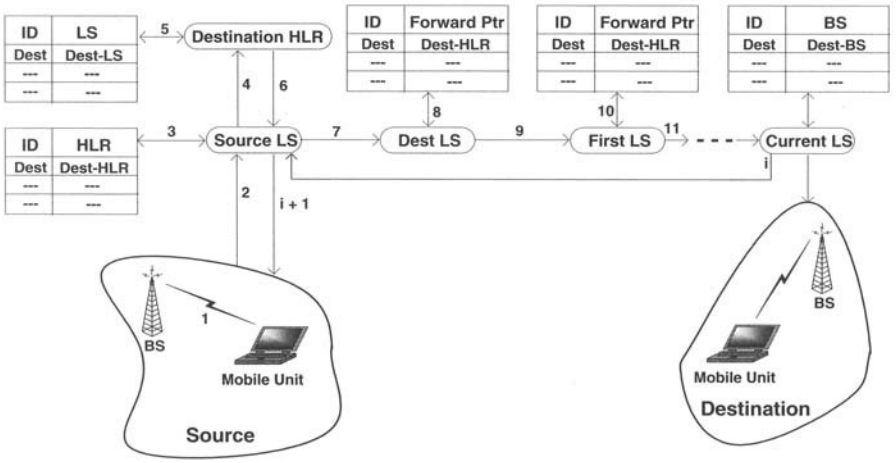

Fig. 3.4 Location search using forward pointer.

Step 2: The source base station sends the query to the Source LS (source location server) for location discovery.

Step 3: Source LS first looks up the VLR to find the location. If the called number is a visitor to the source base station, then the location is known and the connection is set up.

Step 4: If VLR search fails, then the location query is sent to the HLR.

Step 5: The Destination HLR finds the location of destination location server (Dest-LS).

Step 6: The Destination HLR sends the location of destination location server (Dest-LS) to the Source LS.

Step 7: The Source LS finds the first forward pointer (8) and traverses the chain of forward pointers (9, 10, 11, ...) and reaches the Destination location server (Current LS).

Step i: The location of current base station is forward to the Source LS.

ep i + 1: Source LS transfers the address of current base station to the source base station and the call is set up.

Forward Pointer Maintenance: Pointer maintenance is necessary to (a) remove pointers which have not been used for some time and (b) delete dangling pointers. During movement a mobile unit may create a number of pointers including transient loops. In Figure 3.3 when the MU2 returns to R1, the forward pointers R2 $\rightarrow$ R3, R3 $\rightarrow$ R4, and R4 $\rightarrow$ R2 will not be referenced to locate MU2, so they can be safely removed from the search path. The identification of candidates for removal can be achieved in a number of ways. One way is to associate a timestamp with each

forward pointer and define a purge time slots. At a purge slot if *purge slot* > *a pointer timestamp* then this pointer can be a candidate for removal. The another way is to keep a directed graph of pointers. If a loop is found in the graph, then all edges except the last one can be removed. It is possible that in a long path there may be a small loop. For example, in path R2 → R3, R3 → R4, R4 → R3, and R3 → R5, the small loop R3 → R4 and R4 → R3 can be replaced by R3 → R5. In further refinement, path R3 → R5, with R5 being the current location, can be replaced by R2 → R5.

Dangling pointers occur if redundant pointers are not removed in a correct order. In the above removal process, if the path R2 → R3 is removed first, then the path R2 → R5 cannot be set and paths R3 → R4, R4 → R3, and R3 → R5 will create dangling pointers. This is classical pointer management problem with a different effect in mobile scenario.

The entire pointer management process must be synchronized with HLR update. Note that HLR may have been updated many times during the creation of forward pointers. Any reorganization must maintain the location consistency in HLR. Further information about the performance of pointer maintenance schemes can be found in Ref. [8].

3.1.2 Handoff Management

The process of handoff was briefly discussed in Chapter 2. This section discuses how a handoff is managed to provide continuous connectivity. Figure 3.5 illustrates the presence of an overlap region between Cell 1 and Cell 2. A mobile unit may spends some time in this overlap area and the value of this duration depends upon the movement speed of the mobile unit. The duration a mobile unit stays in this area is called the *degradation interval* [10]. The objective is to complete a handoff process while the mobile unit is still in the overlap area. This implies that the handoff must not take more than the *degradation interval* to complete he process. If for some reason the process fails to complete in this area or within *degradation interval*, then the call is dropped.

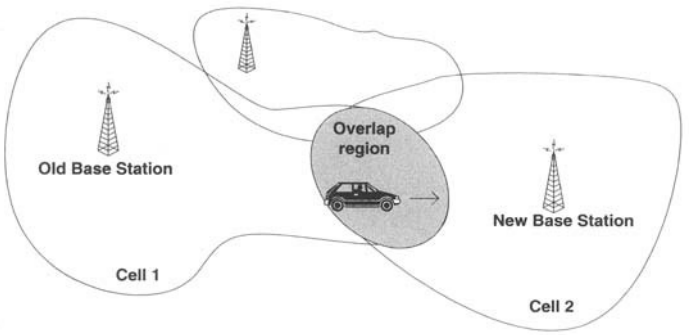

Fig. 3.5 Cell overlap region.

A handoff may happen within or outside a registration area. If it happens within a registration area, then it is referred to as intra-system handoff where the same MSC manages the entire process. An intersystem handoff occurs between two separate registration areas where two MSCs are involved in handoff processing. In each of these cases the handoff processing is completed in three steps:

- **Handoff detection**: The system detects when a handoff process needs to be initiated.
- **Assignment of channels**: During handoff processing the system identifies new channels to be assigned for continuous connectivity.
- **Transfer of radio link**: The identified channels are allocated to the mobile unit.

Handoff Detection

Handoff processing is expensive, so the detection process must correctly detect a genuine and *False Handoff* (see Chapter 2) which also occurs because of signal fading. There are three approaches for detecting handoff effectively and accurately. A brief description of these approaches, which are applied on GSM system but also used in PCS, is presented here and further details can be found in Ref. [10]. They are called:

- Mobile-Assisted Handoff (MAHO)
- Mobile-Controlled Handoff (MCHO)
- Network-Controlled Handoff (NCHO)

Mobile-Assisted Handoff (MAHO): This scheme is implemented in second-generation systems where TDMA technology is used. In this approach, every mobile unit continuously measures the signal strength from surrounding base stations and notifies the strength data to the serving base station. The strength of these signals are analyzed, and a handoff is initiated when the strength of a neighboring base station exceeds the strength of the serving base station. The handoff decision is made jointly by base station and Mobile Switching Center (MSC) or base station controller (BSC). In case the Mobile Unit (MU) moves to a different registration area, an intersystem handoff is initiated.

Mobile-Controlled Handoff (MCHO): In this scheme the Mobile Unit (MU) is responsible for detecting a handoff. The MU continuously monitors the signal strength from neighboring base stations and identifies if a handoff is necessary. If it finds the situation for more than one handoff, then it selects the base station with strongest signal for initiating a handoff.

Network-Controlled Handoff (NCHO): In this scheme, Mobile Unit (MU) does not play any role in handoff detection. The BS monitors the signal strength used by MUs and if it falls below a threshold value, the BS initiates a handoff. In this scheme also BS and MSC are involved in handoff detection. In fact the MSC instructs BSs to monitor the signal strength occasionally, and in collaboration with BSs the handoff situation is detected. The MAHO scheme shares some detection steps of NCHO.

Necessary resources for setting up a call or to process a handoff request may not always be available. For example, during a handoff the destination BS may not have any free channel, the MU is highly mobile and has requested too many handoffs, the system is taking too long to process a handoff, the link transfer suffered some problem, and so on. In any of these cases the handoff is terminated and the mobile unit loses the connection.

Assignment of Channels

One of the objectives of this task is to achieve a high degree of channel utilization and minimize chances of dropping connection due to unavailability of channel. Such failure is always possible in a high traffic area. If a channel is not available, then the call may be blocked (*blocked calls*); and if a channel could not be assigned, then call is terminated (*forced termination*). The objective of a channel allocation scheme is to minimize forced termination. A few schemes are presented here [10].

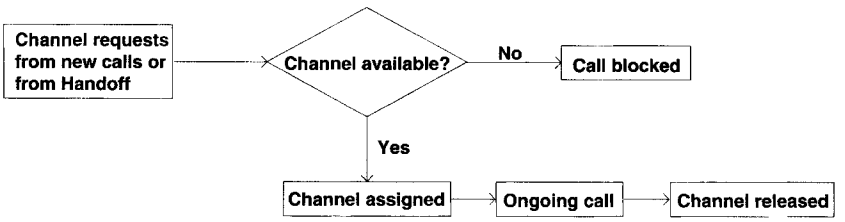

Fig. 3.6 Nonprioritized scheme steps. (Reproduced from Wireless and Mobile Network Architectures under written permission of John Wiley & Sons.)

Nonprioritized Scheme: In this scheme the base station does not make any distinction between the channel request from a new call or from a handoff process. If a free channel is not available then the call is blocked and may subsequently be terminated. Figure 3.6 shows the entire channel assignment process.

Reserved Channel Scheme: In this scheme a set of channels are reserved for allocating to handoff request. If a normal channel is available, then the system assigns it to a handoff request; otherwise the reserved channel is looked for. If no channels are available in either set, the call is blocked and could be dropped. Figure 3.7shows the entire channel assignment process.

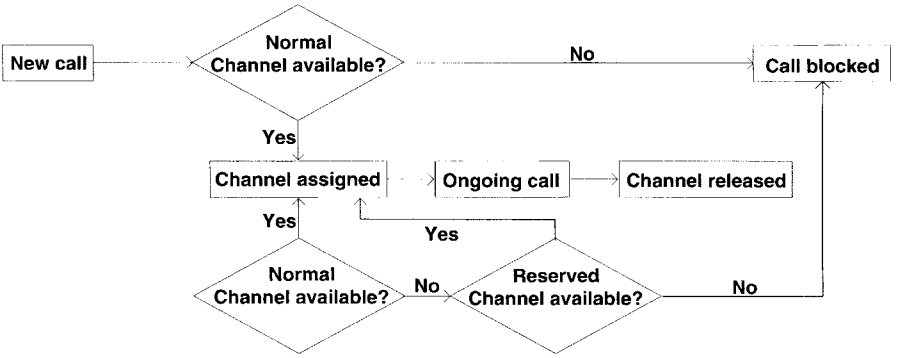

Fig. 3.7 Reserved channel scheme steps. (Reproduced from Wireless and Mobile Network Architectures under written permission of John Wiley & Sons.)

Queuing Priority Scheme: In this scheme a channel is assigned based on some priority. If a channel is available, then the handoff request is process immediately; otherwise the request is rejected and the call is dropped. There is a waiting queue where requests are queued. When a channel becomes available, then one of the requests from the waiting queue is selected for processing. The queuing policy may be *First in First Out (FIFO)* or it may be *measured-based* or some other scheme. In the measured-based approach the request which is close to the end of its degradation interval is assigned a channel first. In the absence of any free channel the call is terminated. Figure 3.8 shows the entire channel assignment process.

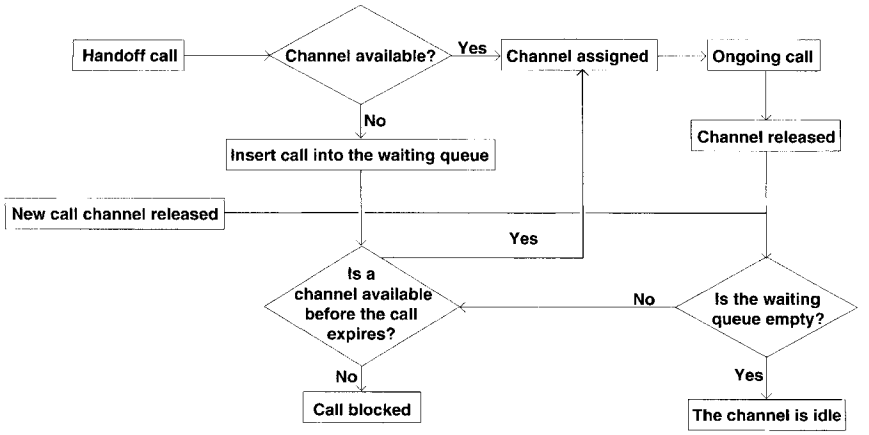

Fig. 3.8 Queuing priority scheme steps. (Reproduced from Wireless and Mobile Network Architectures under written permission of John Wiley & Sons.)

Subrating Scheme: In this scheme a channel in use by another call is subrated, that is, the channel is temporarily divided into two channels with a reduced rate. One channel is used to serve the existing call and the other channel is allocated to a handoff request. Figure 3.9 shows the channel assignment process.

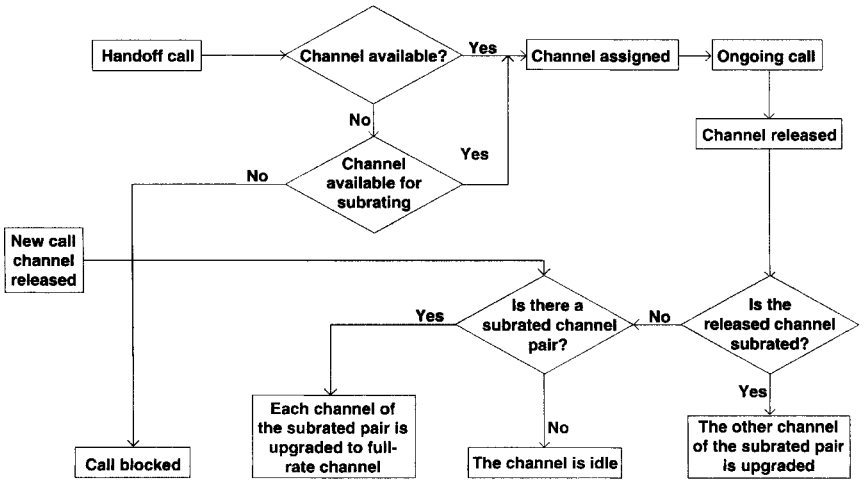

Fig. 3.9 Subrating scheme steps. (Reproduced from Wireless and Mobile Network Architectures under written permission of John Wiley & Sons.)

Radio Link Transfer

The last phase of handoff is the transfer of the radio link. The hierarchical structure of cellular system (PCS and GSM) presents the following five-link transfer cases for which handoff has to be processed.

- **Intracell handoff:** Link or channel transfer occurs for only one BS. In this handoff a MU only switches channel. Figure 3.10 illustrates the scenario.
- **Intercell or Inter-BS handoff:** The link transfer takes place between two BSs which are connected to the same BSC. Figure 3.11 illustrates the scenario.
- **Inter-BSC handoff:** The link transfer takes place between two BSs which are connected to two different BSCs and the BSC is connected to one MSC. Figure 3.12 illustrates the scenario.
- **Intersystem or Inter-MSC handoff:** The link transfer takes place between two BSs which are connected to two different BSCs. These two BSCs are connected to two different MSCs. Figure 3.13 illustrates the situation.

As discussed in Ref. [10], typical call holding time is around 60 seconds. Some real-life data indicates that there could be around 0.5 inter-BS handoff, 0.1 inter-BSC

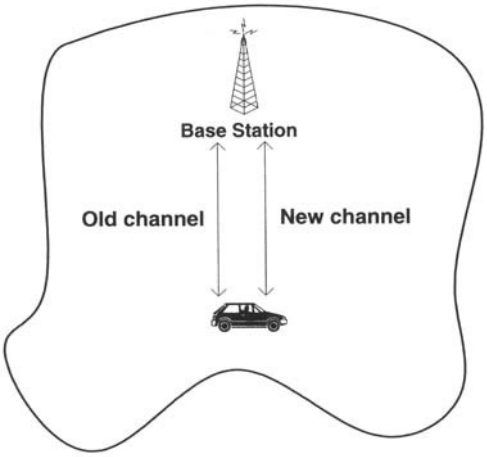

Fig. 3.10 Channel transfer in intracell handoff.

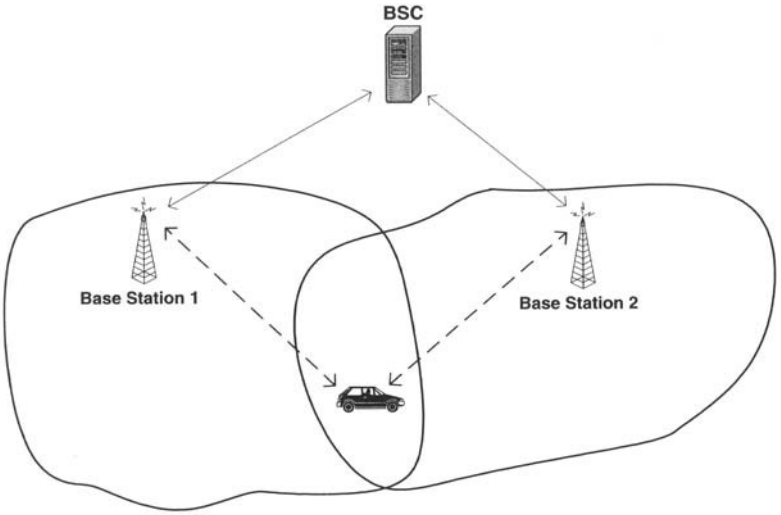

Fig. 3.11 Channel transfer between two BSs with one BSC.

handoff, and 0.05 inter-MSC handoff. The data also indicate that the failure rate of inter-MSC handoff is about five times more than inter-BS handoff. It is quite obvious that efficient processing of handoff is quite important for minimizing the call waiting time.

There are two ways to achieve link transfer. One way is referred to as *Hard Handoff* and the other as *Soft Handoff*.

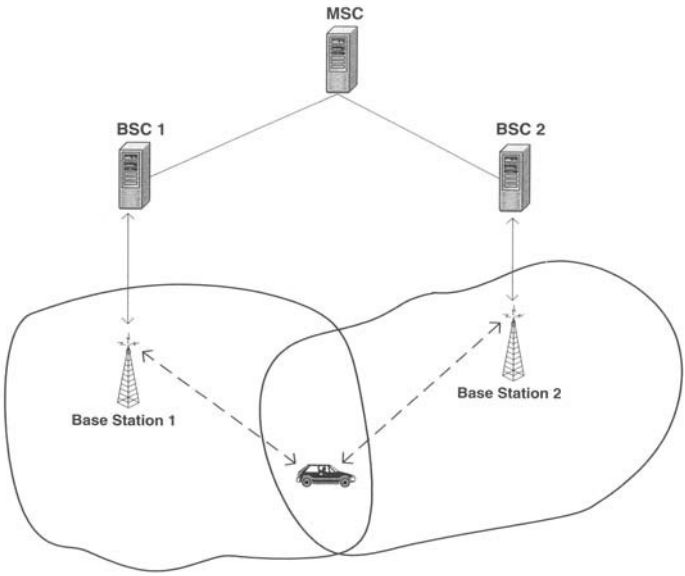

Fig. 3.12 Channel transfer between two BSs connected to two BSCs.

Hard Handoff: In this handoff process the user experiences a brief silence or discontinuity in communication which occurs because at any time the MU is attached to only one BS and when the link is transfer the connection is broken temporarily resulting in a silence. The steps of the handoff for MCHO link transfer is described below. Further detail is given in Ref. [10].

1. MS sends a "link suspend" message to the old BS which temporarily suspends the conversation (occurrence of silence).

2. The MS sends a "handoff request message" to the network through the new BS. The new BS then sends a "handoff acknowledgement" message and marks the slot busy. This message indicates the initiation of the handoff process.

3. This acknowledgment message indicates to MU that the handoff process has started, and so MU returns to the old channel it was using and resumes voice communication while network process the handoff.

4. When the new BS receives the handoff request message, then two cases arise: (a) It is an intra-BS handoff or (b) it is an inter-BS handoff. In the former case the BS sends a handoff acknowledgment message and proceeds with handoff. In the later case, since it is between two different BSCs, the BS must complete some security check. It gets the cypher key from the old BS and associates it with the new channel.

5. The MSC bridges the conversation path and the new BS.

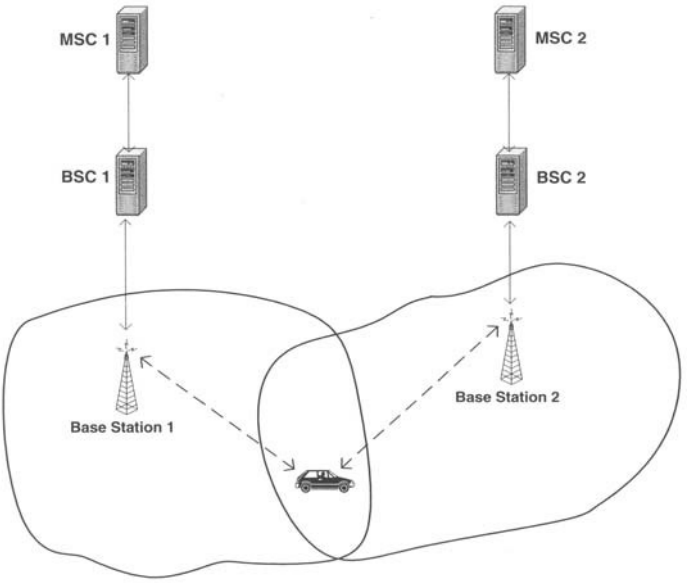

Fig. 3.13 Channel transfer between two BSs with two BSCs connected to two MSCs.

6. On the command of the network, the MS processes the handoff where it releases the old channel by sending an "access release" message to the old BS. In this process the voice communication is briefly interrupted again.

7. The MU sends a "handoff complete" message through the new channel and resumes the voice communication.

A detailed discussion on hard handoff for other kinds of link transfer and soft handoff can be found in Ref. [10].

3.1.3 Roaming

In the presence of multiple wireless service providers the continuous connectivity is provided through *Roaming*. Thus when a mobile moves from one GSM to another system PCS or GSP or some other, the location of MU must be informed by the new service provider to the old service provider. This facility is called roaming facility. These two service providers communicates with each other to complete the location management and the registration process as described earlier.

The other important aspect of roaming is the administrative issues related to billing. Multiple service providers have to come to some agreement about the charges and privileges.

■ **EXAMPLE 6.4**

In electronic commerce applications, such as auctions, it is expected that a typical auction might bring together millions of interested parties. Updates based on bids made must be disseminated promptly and consistently. A mobile system may use broadcast facility to transmit the current state of the auction while allowing the client to communicate their updates using low bandwidth uplink channels. Broadcast based data dissemination is likely to be a major mode of information transfer in mobile computing and wireless environments.

6.2 EFFECT OF MOBILITY ON THE MANAGEMENT OF DATA

The above set of examples illustrates the importance of mobile database systems to manage real-life information processing activities. Mobile systems, however, cannot function without the support of conventional systems. It is, therefore, important to investigate how mobile discipline affects conventional data processing approaches for understanding their seamless integration [3].

In conventional database systems there is one common characteristic: All components, especially the processing units, are stationary. A user must go to a fixed location to use the system. In distributed systems, depending upon the type of data, distribution data may migrate from one node to another, but this migration is deterministic; that is, data move from one fixed source to another fixed destination. Such data migration does not satisfy any mobility criteria.

The integration of geographical mobility is an excellent way to efficiently salvage time wasted in traveling. However, it gives rise to a number of problems related to the maintenance of ACID properties in the presence of personal and terminal mobility. A number of these problems are addressed in the following sections.

The ACID properties of a transaction must be maintained in all data management activities. Concurrency control mechanisms and database recovery schemes make sure that ACID is maintained. In mobile and wireless platform the nature of data processing remains the same, but the situations under which data are processed may change. It is, therefore, important to understand the effect of mobility on data distribution and ACID properties of transactions.

6.2.1 Data Categorization

The data distribution in conventional distributed database systems can be done in three ways: (a) partitioned, (b) partial replication, and (c) full replication. The presence of processor mobility adds another dimension to conventional data distribution. It introduces the concept of *Location-Dependent Data* (LDD).

Location-Dependent Data (LDD): It is a class of data where data values are tightly linked to specific geographical location. There is 1:1 mapping between the data value set and the region it serves. For example, *City Tax* data value is functionally dependent on the city's tax policy. It is possible that all cities may use the same city tax schema, but each city will map to a unique instance of the schema. Some other example of LDD are zip code, telephone area code, etc. In contrast, some classes of data have no association with any location–for example, Social Security Number (SSN), street names, rain fall, snow fall, etc. The value of SSN does not identify any specific location such as a street name. The same street name may exist in Boston or in Seattle or in Kansas City. These are called Location-Independent Data, and the conventional data processing approach interprets all data as location-independent data.

Location Dependent Query: LDD gives rise to *Location-Dependent Query* and *Location-Aware Query*. A location-dependent query needs LDD for computing the result. For example, *What is the distance from the airport to here?* is a location-dependent query because the value of the distance depends on the geographical location of the mobile unit which initiated the query. If the coordinates of the location "here" is not known, then the query cannot be processed. Consider the situation when a person is driving to the airport to catch his flight. He is running late, and so after every 5 minutes he repeats the query *How far is the airport now?* Each answer to this identical query will be different but correct because the geographical location of "here" is continuously changing. A similar situation arises in processing the query *Where am I?* I will continuously ask this query after driving randomly to some location and will have different correct answers. (I may get completely lost but that is a different matter altogether!). This kind of situation exists only when the geographical coordinates of the origin of query continuously change with time. This is a common situation in every day life. If a traveler initiates a query *What is the sales tax of this city?* while passing through a city, then the answer must be related to the current city and not to the next city where he arrives soon after initiating the query. A similar situation arises in listening to a radio station while traveling. When the traveler crosses the broadcast boundary, the same frequency tunes to a different radio station and the broadcast program changes completely.

In processing a location-dependent query, the necessary LDD and the geographical location of the origin of the query must be known. This requires that the system must map the location with the data to obtain correct LDD. A number of service providers have location discovery facility which can be used to access LDD.

Location-Aware Query: This type of query includes reference to a particular location either by name or by suitable geographical coordinates. For example, *What is the distance between Dallas and Kansas City?* is a location-aware query because it refers to locations Kansas City and Dallas. The answer to this query or any location aware query does not depend on the geographical location of the query; as a result, the mobility does not affect its processing.

6.2.2 Location Dependent Data Distribution

The 1:1 mapping between data and its geographical location restricts the three data distribution approaches. The horizontal fragmentation and vertical fragmentation of a relational database must include the location information implicitly or explicitly. The partition of database, however, becomes easier because the decision is solely based on the location parameter. The concept of *data region* is helpful to understand the distribution of database partitions in mobile databases.

Definition 6.1 *A data region is a geographical region or a geographical cell, and every geographical point of this region satisfies 1:1 mapping with data.*

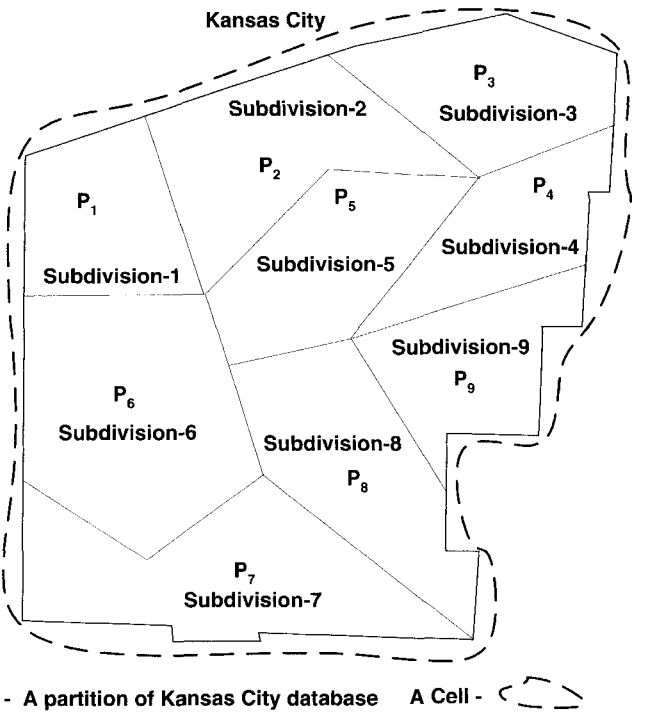

Fig. 6.1 Database partition for LDD.

Figure 6.1 illustrates data distribution for data partition scheme. It assumes Kansas City as a data region for city sales tax. The entire data region is enclosed in a cell. Every location of Kansas City satisfies 1:1 mapping between city tax value and the location. The entire Kansas City database is partitioned into subdivisions identified by P_1 through P_9. All subdivisions map to the same city sales tax; as a result, all subdivisions charge the same city tax. If every subdivision maintains its own database, then at each subdivision a database partition can be stored. A mobile unit which moves among subdivisions will see the same one consistent value of a data item.

■ **EXAMPLE 6.5**

A hotel chain or franchise can be used to demonstrate the problem of data replication and its consistency for mobile databases. A particular hotel has a number of branches across the nation. Each branch offers identical services; however, its room rent, policy, facilities, etc., would depend on the branch location. Thus, the same-size suite may cost more in Dallas than in Kansas City. The data consistency constraints in Kansas City might be different from those in Dallas, because of local taxes and environment policies. Each branch may share the same schema but their instantiations (values for the data) may differ.

In a partial replication approach the same partition can be replicated at more than one subdivision. For example, at subdivision 1 and subdivision 2, P_1 and P_2 can be replicated without affecting the consistency. In a full replication, also the entire database can be replicated and used at all subdivisions in a consistent manner.

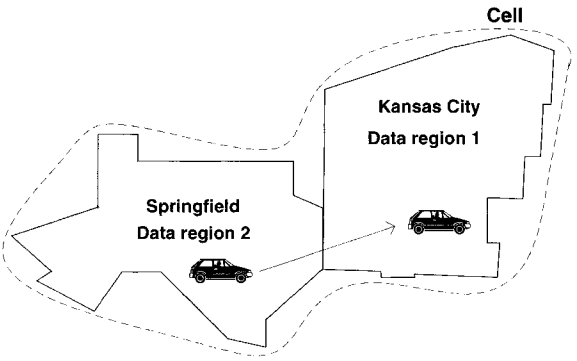

Fig. 6.2 Database replication restriction.

The situation does not change if the data region is covered by multiple cells. A mobile unit can move from one subdivision to another and use the same data item in both subdivisions. However, the situation changes when a cell covers two or more data regions as shown in Figure 6.2. Data of one region cannot be replicated at another region. For example, the sales tax rate of Kansas City (region 1) cannot be replicated at Springfield (region 2). This constraint requires that a location-dependent query in Springfield must be processed in Springfield before the client enters Kansas City. This restriction also affects mobile data caching. A mobile unit must clear its cache before entering to another data region for maintaining global consistency.

Since the distribution of LDD is dependent on geographical locations, its distribution is defined as *spatial distribution* to distinguish it from the conventional distribution which is called a *temporal distribution*. In spatial distribution and in temporal distribution *spatial replication* and *temporal replication*, respectively; are used.

Definition 6.2 *Spatial Replication refers to copies of data objects which may have different correct data values at any point in time. Each value is correct within a given location area. One of these copies is called a* **Spatial Replica.**

Definition 6.3 *Temporal Replication refers to copies of data objects all of which have only one consistent data value at any point in time. One of these copies is called a* **Temporal Replica.**

The temporal distribution mainly considers local availability of data and the cost of communication, but for spatial distribution the geographical location must also be included. The identification of data as spatial and temporal affects the definition of consistency.

Effect of Mobility on Atomicity: The property of atomicity guarantees that partial results of a transaction do not exist in the database. If a transaction fails to commit, then all its effects are removed from the database. The mobility does not alter the definition of atomicity but makes its enforcement quite difficult. Transaction execution log is required for implementing atomicity. In a conventional system the log is stored at the server and is easily available. In a mobile system, conventional logging approach does not work satisfactorily because a mobile unit gets connected and disconnected to several servers when it is mobile. There are a number of ways to manage a transaction log in mobile systems; this is discussed in the recovery section.

Effect of Mobility on Consistency: In a centralized or distributed environment there is only one correct value for each data object. The term *mutual consistency* is used to indicate that all values of the same data item converge to this one correct value [2]. A replicated database is said to be in a *mutually consistent state* if all copies have the exact same value [2]. In addition, a database is said to be in a *consistent state* if all integrity constraints identified for the database are followed [2].

In a mobile database system the presence of location-dependent data defines two types of consistency: *Spatial Consistency* and *Temporal Consistency.*

Definition 6.4 *Spatial consistency indicates that all data item values of a spatial replication are associated with one and only one data region, and they satisfy consistency constraints as defined by the region. Thus there is 1:1 mapping between data value and the region it serves.*

Every mobile unit that initiates transactions in a region must get a consistent view of the region and the database must guarantee that the effect of the execution of the transactions is durable in that region. To achieve this state, the region must satisfy temporal consistency as well.

Definition 6.5 *Temporal consistency indicates that all data item values must satisfy a given set of integrity constraints. A database is temporally consistent if all temporal replicas (replication of data items at multiple sites) of a data item have the same value.*

Effect of Mobility on Isolation: Transaction isolation ensures that a transaction does not interfere with the execution of another transaction. Isolation is normally enforced by some concurrency control mechanism. As with atomicity, isolation is needed to ensure that consistency is preserved.

In mobile database systems a mobile unit may visit multiple data regions and process location-dependent data. The important thing is to ensure that execution fragments satisfy isolation at the execution fragment level. It will do so under some concurrency control mechanism which must recognize the relationship between a data item. The mechanism must enforce isolation in each region separately but achieve isolation for the entire transaction. This is quite different from a conventional distributed database system which, does not recognize spatial replication and thus does not enforce regional isolation.

Effect of Mobility on Durability: Durability guarantees the persistence of committed data items in the database. In mobile database systems the durability is regional as well as global. For spatial replicas and temporal replicas; regional durability and global durability, respectively are enforced.

Effect of Mobility on Commit: Transaction commitment is not affected by mobility; however, because of the presence of location-dependent data, a *location commit* is defined. A location commit binds a transaction commit to a region. For example, a department manager initiates the following transaction on his mobile unit: *Reserve 5 seats in a vegetarian restaurant located 1 mile from here.* . This is a location-dependent update transaction, and it must be processed in the region where the restaurant is located. The confirmation must be sent back as fast as possible to the manager, which becomes necessary if the manager is waiting for the confirmation. The database server responsible for processing this transaction must first map the location of the query and the location of the restaurant and then access the correct database for making the reservation. The entire execution remains confined to the region until the transaction commits. Thus the process of commit is identical to the conventional notion of transaction commit; however, the requirements for the commit are different. It is called *location-dependent commit* to differentiate it from conventional notion of commit.

Definition 6.6 *An execution fragment,* e_i*, satisfies a* **Location-Dependent Commit** *iff the fragment operations terminate with a commit operation and a location to data mapping exists. Thus all operations in* e_i *operate on spatial replicas defined on the location identified by location mapping. The commit is thus associated with a unique location* L*.*

Effect of Connectivity on Transaction Processing

In a mobile environment an MU can process its workload in a *continuously connected* mode or in *disconnected* mode or in an *intermittent connected* mode.

Connectivity mode: In this mode an MU is continuously connected to the database server. It has the option of caching required data for improving performance or can request data from the server any time during transaction processing. If necessary, it can enter into doze mode to save power and becomes active again. However, this mode is expensive to maintain and is not necessary for processing users workload.

Disconnected mode: In this mode an MU voluntarily disconnects from the server after refreshing the cache and continues to process workload locally. At a fixed time it connects and sends its entire cache to the server using wireless or wired link. The server install the contents of the cache such a way that global consistency is maintained.

Intermittent connected: This mode is similar to the disconnected mode, but here the MU can be disconnected any time by the system or voluntarily by the user. The disconnection by system may be due to lack of channel, low battery, security, etc. The user may disconnect the MU to save power or to process data locally, or no communication with the server is required for some time. Unlike disconnected mode, intermittent mode does not have any fixed time for connecting and disconnecting an MU.

This type of connectivity is useful for agents dealing with customers–for example, insurance agents, UPS or FedEx, postal delivery, etc. For postal delivery, the entire day's delivery can be defined as a long workflow. The agent delivers a packet to a house and locally updates the cached database on the mobile device. At the end of the day or at a prescribed time the agent connects and ships the entire cache to the server with through a wired or wireless channel. It is possible that the agent may connect to server to report the status of the high priority shipment. The stock evaluator in a supermarket also works in a similar manner. After recording the stock level the agent connects the server for updating the main database. Connection on demand is also a form of intermittent connectivity because a user's need for data is usually unpredictable.

The database consistency in disconnected or intermittently connected mode is hard to define and maintain. This becomes relatively difficult in an e-commerce or m-commerce environment, which can be explained with a simple example. Consider a company called Pani[1] Inc., sells water purifier aggressively. Two agents Kumar and Prasad go house to house in a subdivision, demonstrate the water filter, and try to win household's business. Suppose the company has 100 water purifier units in the warehouse and wants to sell them aggressively. Pani Inc. does not want to take a chance, so it asks each agent to download 100 units on their laptop and to sell them in a day. In this way if each agent sells 50 units, then the job is done. Now suppose with a bit of luck and with some persuasion, Kumar and Prasad both sell 100 units without being aware of each others success. This pushes the database into a real mess. Pani Inc. handles the situation using a "back order" scheme and reduction in the cost

[1]Pani is a Hindi word which means water

of water purifier. So in this situation, how can we define the consistent state of the database? One way could be "existing inventory + back order," but this is quite risky. If Pani Inc. could not supply all back orders within the promised time, then some orders may have to be rolled back; as a result, it may be difficult to maintain ACID constraints.

Managing ACID transactions processing in the connected state is easy and can be handled in a conventional manner. However, their processing in the disconnected and intermittent connected modes requires new caching consistency approaches, new locking approaches, new commit protocol, new rollback and abort schemes, and most important a new transaction model or new way of processing ACID transactions.

6.3 SUMMARY

This chapter discussed the relationship between mobility and transaction processing. A clear understanding of this relationship is necessary for the development of mobile transaction model and its management. Three types of connectivity modes and their effect on database consistency and transaction processing were explained. In the next chapter, various ways of executing ACID transactions on mobile database system and mobile transaction models are presented.

Exercises

1. Define processor mobility from data management and transaction execution viewpoints. Identify the set of problems exclusive to each.

2. In the presence of processor mobility, data and transactions acquire exclusive properties. Identify and explain these properties. How do they affect database query processing?

3. Explain the difference between location-dependent, location-independent, and location-free queries. Give at least two real-life examples of each of them.

4. Explain the problems of location-dependent data distribution. How do they affect database integrity and consistency. Are they similar to problems of data distribution in federated and multidatabase systems? Explain your answer.

5. Give your own thoughts on the effect of mobility on database consistency, database integrity, database distribution, and transaction execution.

REFERENCES

1. D. Barbara, "Mobile Computing and Databases - A survey," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 1, January 1999.

2. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Engleewood Cliffs, NJ, 1999.

3. E. Pitroura and B. Bhargava, "Revising Transaction Concepts for Mobile Computing," in *Proceedings of Workshop on Mobile Computing Systems and Application*, 1994.

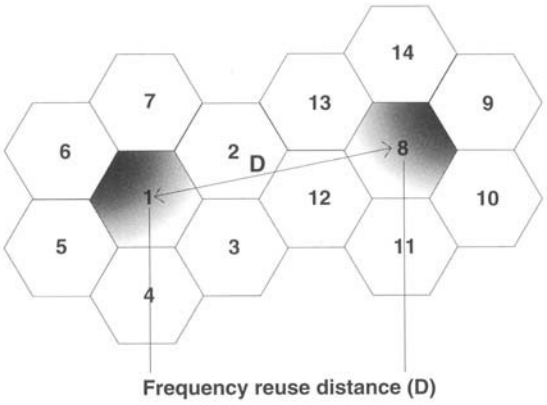

Fig. 7.4 Adjacent and nonadjacent cells.

7.3 MOBILE TRANSACTION MODEL

This section discusses the need for a new transaction model for mobile database systems. It was recognized in some of the earlier chapters that the conventional ACID transaction model was unable to satisfactorily manage mobile data processing tasks. Some of the important reasons were: the presence of *handoff*, which is unpredictable; the presence of *doze mode*, *disconnected mode*, and *forced disconnection*; lack of necessary resources such as memory, and wireless channels; presence of location-dependent data; etc. To manage data processing in the presence of these new issues, a more powerful transaction model or ACID transaction execution model that can handle mobility during data processing was highly desirable. Two approaches to manage mobile databases were proposed, and the chapter discusses them in detail. Under each approach, a number of schemes were developed and each approach addressed some specific issues.

The entire topic of mobile transaction modeling is highly research-oriented; and although significant number of schemes have been proposed, none has become a commonly accepted method. The chapter, therefore, identifies each execution model and presents the original scheme as described by the author(s) of the report. The discussion clearly indicates the incremental understanding of mobile data processing and how researchers addressed related issues with their execution models.

In Chapter 6 the effect of new parameters such as mobility, location information, intermittent connectivity, etc., was investigated. It was illustrated that the basic ACID transaction model was unable to handle mobility aspect and location dependent processing, which are now quite common in transactional requests.

There are basically two ways to handle transactional requests on MDS: (a) execution model based on ACID transaction framework and (b) mobile transaction model and its execution. The first approach creates an execution model based on ACID transaction framework. In the second approach a user query is mapped to a mo-

bile transaction model and executed under mobile ACID constraints. The execution model approach managed to handle mobility and location information, but its scope was somewhat limited. This gave rise to the development of mobile transaction models which captured and assimilated mobility and location property in its structure. These two approaches are discussed in detail in subsequent sections.

7.4 EXECUTION MODEL BASED ON ACID TRANSACTION FRAMEWORK

The concept of ACID transaction was introduced for consistency-preserving database processing. Informally, *"A transaction is a collection of operations on the physical and abstract application state"* [11]. The conventional transaction model makes sure that the ACID properties of a transaction are maintained during database processing [11]. The introduction of mobility significantly changed the database architecture and management paradigm, and it became clear that the strict enforcement of ACID properties was not necessary to maintain database consistency. As a matter of fact, mobility changed and in many cases had to relaxed the notion of consistency because in mobile database systems the notion of consistency is closely related to locations in the geographical domain, which is defined as follows:

Definition 7.1 *The Geographic Domain, G, is the total geographical area covered by all mobile units of a cellular system. Thus,* $G = (C_1 + C_2 + \cdots + C_n)$*, where* C_i *represent the area of a cell.*

Definition 7.2 *A Location is a precise point within the Geographic Domain. It represents the smallest identifiable position in the domain. Each location is identified by a specific id, L. Also,* $G = \cup L, \forall L$ *and* $C_i = \{L_i, L_2, ..., L_m\}$.

In reality, a location of a mobile unit is identified with reference to the BS. If the geographic domain were on the Earth, then one can think of a location as a *latitude/longitude* pair. However, the granularity of the location used may be larger. For example, the location could be an address, city, county, state, or country.

It is important to understand the complex relationship among the data, the operations to be performed on the data, and the termination of the execution for the development of an execution model. These issues were introduced in Chapter 6 and are further elaborated in this chapter.

Location Dependent Query - LDQ: In legacy systems, the frequency of access of a data items and not their association with geographical locations is used in data distribution (partition and partial replication). In MDS this association plays an important role in their processing as well as in their distribution. Figure 7.5 identifies some important points. Suppose a person is traveling by car from Dallas to Kansas City and asks, *Tax rate please?* The answer to this query will depend on where actually the query originated. If the location is Dallas, then it will give the tax rate for Dallas; and if it is Kansas City, then the tax rate will be for Kansas City. Now

d. After T_i ends its execution, the pre-commit phase starts. In pre-commit if there is no conflict, then *pre-write* lock mode is converted to write lock mode (Table 7.2).

e. The pre-commit of T_i is announced where all read locks are released.

f. The final commit begins where the database is updated with pre-write values (final write). All write locks are released and commit is announced.

7.5.1 Pre-write Execution in Mobile Database Systems

The pre-write execution model tries to work within resource constraints of mobile database system. One of the important consideration is the speed of the CPU. For slow CPUs the execution model does not scale very well and MUs act as a simple client with no database processing capability. The following transaction execution example assumes a high speed CPU.

MUs with high-speed CPUs store consistent data items in their local cache. When a transaction arrives at an MU, it uses cached data to process reads and returns the pre-write values. Those reads for which cache does not have data are sent to the server for processing. The server returns pre-write values or write values. The transaction pre-commits when all values are returned by the server and MU has also completed its processing. All locks are released and the pre-commit schedule is sent to the server for the final commit of the transaction.

In some data processing situations, tolerable difference between a *pre-write* version and a final write version may appear. Consider a transaction T_i that executes a *pre-read* on data item x which is the result of a *pre-write* of T_j. T_j commits at the server. In some cases, T_i may have an old value of x, but it is tolerated–for example, draft and final version of a graphic object or some average salary data. A minor difference in this version is not likely to influence the decision outcome.

7.6 MOBILE TRANSACTION MODEL

In the last few sections, mobile execution models for ACID transactions were discussed in detail. An execution model provides a scheme to execute ACID transactions in a resource-limited mobile platforms; however, they have some inherent limitations. Later mobile transaction models were developed to take care of these limitation. A number of such transaction models are discussed in this section.

7.6.1 HiCoMo: High Commit Mobile Transaction Model

This model was presented in Ref. [29]. Although it has been presented as a mobile transaction model, in reality it is a mobile transaction execution model. The execution model is mainly for processing aggregate data stored in a data warehouse which resides in mobile units. Since the data warehouse resides in mobile units, *HiCoMo*

transactions are always initiated on mobile units where they are processed in a disconnected mode. As a result, transaction commitments are quite fast. The results of these transactions are then installed in the database upon reconnection.

The base database resides on the fixed network. It is manipulated by transactions called *base* or *source* transactions. These transactions are initiated at the fixed network. Transaction which are initiated and processed at mobile units are called *HiCoMo*. Since HiCoMo transactions do specialized processing, it is based on the following assumptions:

- The data warehouse stores aggregate data of the following types: average, sum, minimum, and maximum.
- Operations such as *subtraction*, *addition*, and *multiplication* are allowed with some constraints on their order of application.
- The model allows some margin of errors. This margin can be defined before allowed operations are initiated and their value can be varied between a lower and an upper bound.

The structure of *HiCoMo* transaction is based on *nested* transaction model. The database consistency is satisfied through *convergence* criteria. It is satisfied when the states of the base database and the data warehouse in mobile units are identical. This transaction model ensures that convergence is always satisfied.

As mentioned earlier, the base database at the server is updated by *source* transactions. This requires that to install updates of *HiCoMo* transactions, they must be converted to *source* transactions. This conversion is done by a *Transaction Transformation Function*, which works as follows:

- **Conflict detection:** A conflict is identified among other *HiCoMo* transactions and between *HiCoMo* and *bases* transactions. If there is a conflict between *HiCoMo* transaction, then the transaction which is being considered for transformation is aborted.
- **Base transaction generation:** In the absence of a conflict, initial *base* transactions are generated and executed as subtransactions on the base database at the server. The type of base transaction depends upon the *HiCoMo* transactions.
- **Alternate base transaction generation:** It is possible that some of these subtransactions may violate integrity constraints (may be outside the error margin) and, therefore, are aborted. These updates are tried again by redistribution of error margin. In the worst-case scenario the original *HiCoMo* transactions are aborted. If there is no integrity violation, then *base* transactions are committed.

7.6.2 Moflex Transaction Model

A mobile transaction model called *Moflex*, which is based on a *flexible* transaction model [13], is presented in Ref. [24]. The structure of a *Moflex* has 7 components and can be defined as

Moflex transaction T = {*M, S, F, D, H, J, G*}

M = $\{t_1, t_2, \ldots, t_n\}$, where t_i are compensable on noncompensable subtransactions. Every compensable t_i is associated with a corresponding compensating transaction.

S = a set of success-dependencies between t_i and t_j ($i \neq j$). This defines the serial execution order of these subtransactions. Thus, t_j has a success-dependency on t_i (i.e., $t_i <_s t_j$) if t_j can be executed only after t_i commits successfully.

F = a set of failure-dependencies which indicates that t_j can be executed only after t_i has failed. This dependency is represented as (i.e., $t_i <_f t_j$).

D = a set of external-dependencies which indicates that t_i can be executed only if it satisfies predefined external predicates. These predicates are defined on time (p), cost (Q), and location (L).

H = a set of handoff control rules which manages the execution of subtransactions in the presence of a handoff. In this event a subtransaction may *continue* its execution or *restart* or *split-resume* or *split-restart*. These execution states or modes are related to handoff and are explained later.

J = a set of acceptable join rules which are used to determine he correct execution of a subtransaction.

G = a set of all acceptable states of T (*Moflex*).

A *Moflex* transaction can be (a) not submitted for execution – N , (b) currently under execution – E, (c) successfully completed – S or (d) failed – F. An execution of T is regarded as being complete if its current state exists in set G. When this is satisfied, then T can commit. Otherwise, if no subtransaction of T is executing or can be scheduled for execution, then T is aborted.

It is possible to process a location-dependent query with *Moflex*. The location-dependent predicate, along with other constraints, can be defined in terms of time such as from 8 AM to 5 PM. For example, a temporal dependency, which is a member of D, can be stated as follows:

$D = \{P, Q, L\}$
$P = \{8 < time\ (t_1) < 17,\ 8 < time\ (t_2) < 17\}$
$Q = \{cost\ (t_2) < \$100,\ cost\ (t_3) < \$100\}$
$\{t_1, t_4\}$

When a handoff occurs during the execution of T, then the subtransaction can further split into finer subtransactions. If the parent subtransaction is compensable and processing location-dependent data, then the handoff rule forces the subtransaction to abort and restart in the new cell. A restart can be *split-restart* where the value of the partial execution of the subtransaction in the last cell is preserved. In the case of location-independent subtransaction, it further splits into finer subtransactions. One

of these subtransactions which represents the portion of execution occurring in the last cell is free to commit.

An Example of a Moflex

An emergency patient dispatch query can be stated as follows. The objective of this hypothetical transaction is to illustrate how the transaction fits into *Moflex* transaction structure. *Find the right hospital or take the patient the default hospital, then dispatch patient status to the emergency doctor for getting the correct treatment.* This can be expressed in a *Moflex* as

$M = \{t_1(\text{C}), t_2(\text{C}), t_3(\text{NC}), t_4(\text{C}), t_5(\text{C})\}$
$S = \{t_1 <_s t_3, t_2 <_s t_3, t_1 <_s t_4\}$
$D = \{t_1, t_4\}$
$H = \{\text{restart}(t_1), \text{continue}(t_2), \text{continue}(t_3), \text{split-resume}(t_4), \text{continue}(t_5)\}$
$J = \{\text{user}(t_4)\}$
$G = \{(\text{S}, \text{-}, \text{S}, \text{S}, \text{S}), (\text{-}, \text{S}, \text{S}, \text{-}, \text{S})\}$

In this example in set G, S indicates a successful execution of *Moflex* and "-" means that the execution state of the subtransaction does not have to be one of the predefined states. Further details about *Moflex* can be found in Ref. [24].

7.6.3 Kangaroo Mobile Transaction Model

In Ref. [7] a transaction model called *Kangaroo* is presented which captured both data and the movement of mobile units. The model is based on a split transaction model and enforces the majority of ACID properties.

A global or parent Kangaroo transaction, KT, is composed of a number of subtransactions. Each subtransaction is similar to an ACID transaction, which is composed of a set of reads and writes. These subtransactions are called *Joey Transaction (JT)* and are local to a base station. Upon initiation of a Kangaroo transaction, a base station creates a JT for its execution which may be executed at mobile units. When these mobile units migrate to another cell, the base station of this cell takes control of the execution of this transaction.

KTs support transaction execution in *Compensating* or *Split* modes. When a failure occurs in a compensating mode, the JT all execution (preceding or following) is undone and previously committed JTs are compensated. It is difficult for the system to identify a compensating mode, so users provide useful input for creating compensating JTs. The default execution mode is *split* mode. When a failure occurs (when a JT fails) in a default mode, then no new local or global transaction is created from KT and previously committed JTs are not compensated. As a result, in compensating mode, JTs are serializable but may not be in split mode.

Kangaroo transaction processing: A KT, when initiated by a mobile unit, is assigned a unique identity. The initial base station immediately creates a JT with a unique identity and becomes responsible for its execution. There is one JT per base station. When the mobile unit encounters a handoff (i.e., moves to a different cell), KT is split into two transactions – JT1 and JT2. Thus the mobility of a mobile unit is captured by splitting a KT into multiple JTs. These JTs are executed sequentially; that is, all subtransactions of JT1 are executed and committed before all subtranactions of JT2. Further details on KT and be found in the original paper.

Some other models have been reported in the literature which are mentioned briefly here. The semantics-based mobile transaction processing scheme [57] views mobile transaction processing as a concurrency and cache coherency problem. The model assumes a mobile transaction to be a long-lived, one characterized by long network delays and unpredictable disconnections. This approach utilizes the object organization to split large and complex objects into smaller, manageable fragments. A stationary database server dishes out the fragments of a object on a request from a mobile unit. On completion of the transaction the mobile hosts return the fragments to the server. These fragments are put together again by the merge operation at the server. If the fragments can be recombined in any order, then the objects are termed *reorderable* objects. Since a single database server is assumed, the ACID properties can be maintained.

7.6.4 MDSTPM Transaction Execution Model

An execution model called *Multidatabase Transaction Processing Manager (MDSTPM)* is reported in Ref. [56] which supports transaction initiation from mobile units. The model uses message and queuing facilities to establish necessary communication among mobile and stationary (base station) units. At each stationary unit a personal copy of MDSTPM exists which coordinates the connected and disconnected execution of transactions submitted at mobile units.

The MDSTPM has the following components:

- **Global Communication Manager (GCM):** This module manages message communication among transaction processing units. It maintains a message queue for handling this task.

- **Global Transaction Manager (GTM):** This module coordinates the initiation of transactions and their subtransactions. It acts as a *Global Scheduling Submanager (GSS)* which schedules global transactions and subtranactions. It can also act as a *Global Concurrency Submanager (GCS)*, which is responsible for the execution of these transactions and subtranactions.

- **Local Transaction Manager (LTM):** This module is responsible local transaction execution and database recovery.

- **Global Recovery Manager (GRM):** This module is responsible for managing global transaction commit and their recovery in the event of failure.

- **Global Interface Manager (GIM):** This serves as a link between MDSTPM and local database managers.

These transaction models did address most of the important issues of mobility, however, no single model captured or incorporated these issues at one place. In the Kangaroo model a transaction issued by a user at one mobile unit can be fragmented and executed at multiple mobile units. This is acceptable on the research level, but in reality this does not happen. A mobile unit is a resources dedicated to its own transactions and not open for execution sharing. The location-dependent, location-aware, location-independent, intermittent-execution, etc., are some of the important issues which are interrelated and need a unified processing by a single model. A model called *Mobilaction* has tried to capture these into one model which is discussed next.

7.6.5 Mobilaction–A Mobile Transaction Model

In this section a new mobile transaction model called *Mobilaction* is presented in Ref. [20]. Mobilaction is capable of processing location-dependent data in the presence of spatial replication. It is composed of a set of subtransactions, which is also called *Execution Fragments*, and each fragment is a Mobilaction.

Mobilaction is based on the framework of the ACID model. To manage location-based processing, a new fundamental property called "location (L)" is incorporated extending the ACID model to ACIDL. The "location (L)" property is managed by a location mapping function.

Definition 7.8 *Fragment Location Mapping FLM: Each execution fragment, e_j, of a mobile transaction, T_i, is associated with a unique location. Given a set of execution fragments E, FLM is a mapping $FLM : E \rightarrow L$.*

The *FLM* identifies (a) the correct geographical location and (b) the correct spatial replica (LDD) for the execution of a fragment. In addition, it is used to ensure spatial consistency of fragments within a transaction. We first explain how Mobilaction satisfies ACID properties and then formally define Mobilaction.

7.6.6 Atomicity for Mobilaction

The purpose of atomicity is to ensure the consistency of the data. However, in a mobile environment we have two types of consistency. Certainly, atomicity at the execution fragment level is needed to ensure spatial consistency. However, transaction atomicity is not. We could have some fragments execute and others not.

Definition 7.9 *A mobile transaction, T_i, satisfies Spatial Atomicity iff each execution fragment, e_ij, of T_i is atomic. T_i is said to be Spatially Atomic iff each execution fragment, e_ij, is atomic.*

7.6.7 Isolation for Mobilaction

Transaction isolation ensures that a transaction does not interfere with the execution of another transaction. Isolation is normally enforced by some concurrency control

mechanism. As with atomicity, isolation is needed to ensure that consistency is preserved. Thus we need to reevaluate isolation when spatial consistency is present. As with consistency, isolation at the transaction level is too strict. The important thing is to ensure that execution fragments satisfy isolation at the execution fragment level.

Definition 7.10 *A mobile transaction, T_i, satisfies* **Spatial Isolation** *iff each execution fragment, e_ij, of T_i is isolated from all execution fragments of T_i or any other transaction.*

Note that Mobilaction will need to implement a concurrency control technique at the fragment level. Any concurrency control technique could be used. As a matter of fact, a different technique could be used for each fragment.

7.6.8 Consistency and Durability for Mobilaction

A conventional transaction commit satisfies the durability property. There is normally only one commit operation per T_i. However, to ensure spatial consistency, spatial isolation, and spatial atomicity, the mobility property requires that the commit of Mobilaction must also change. We introduce the concept of location-dependent commit.

Definition 7.11 *An execution fragment, e_ij, satisfies a Location-Dependent Commit iff the fragment operations terminate with a commit operation and a FLM exists. Thus all operations in e_ij operate on spatial replicas defined by a data region mapping on the location identified by the FLM. The commit is thus associated with a unique location, L.*

Definition 7.12 *An* **Execution Fragment** *e_ij is a partial order $e_ij = \{\sigma_j, \leq_j\}$, where*

- *$\sigma_j = OS_j \cup \{N_j\}$, where $OS_j = \cup_k O_jk$, $O_jk \in \{read, write\}$, and $N_j \in \{abort_L, commit_L\}$. Here these are a location-dependent commit and abort.*
- *For any O_jk and O_jl where $O_jk = R(x)$ and $O_jl = W(x)$ for a data object x, then either $O_jk \leq_j O_jl$ or $O_jl \leq_j O_jk$.*
- *$\forall O_jk \in OS_j, O_jk \leq_j N_j$.*

The only difference between an execution fragment and a transaction is that either a location dependent commit or abort is present instead of a traditional commit or abort. Every fragment is thus associated with a location. However, keep in mind that if the data object being updated is a temporal replica, then the fragment updates all replicas. Thus it is not subjected to location constraints and appears as a regular transaction.

Definition 7.13 *A Mobilaction (T_i) = $<F_i, L_i, FLM_i>$, where $F_i = \{e_i1, ..., e_in\}$ is a set of execution fragments, $L_i = \{l_i1, ..., l_in\}$ is a set of locations, and $FLM_i =$*

$\{flm_i1, ..., flm_in\}$ is a set of fragment location mappings, where $\forall j$, $flm_ij(e_ij) = l_ij$. In addition, $\forall j, k, l_ij <> l_ik$.

In traditional database systems, ACID transaction is assumed to be a unit of consistency. Even with spatial atomicity, this is still the case with a Mobilaction. A Mobilaction is a unit of consistency. That is, given a database state which is both temporally and spatially consistent, a Mobilaction T_i converts this state into another temporally and spatially consistent state.

Table 7.3 Summary of previous mobile transaction models and ACID adherence

Model	A	C	I	D	Request	Execute
Kangaroo	No	No	No	No	MU	Fixed Network
Reporting	No	No	No	No	N/A	N/A
Co-transaction	No	No	No	No	N/A	N/A
Clustering	No	No	No	No	MU	MU or Fixed Network
Semantics	Yes	Yes	Yes	Yes	MU	Restricted Server/MU
MDSTPM	No	No	No	No	MU	MU or Fixed Network

Table 7.3 compares the various mobile transaction models based on ACID property compliance and processing location. Due to the fact that the Kangaroo model assumes the autonomy of the underlying DBMS systems, subtransactions are allowed to commit/abort independently. Atomicity may be achieved if compensating transactions are provided. While the Semantics approach allows processing anywhere in the mobile platform, it is a restricted type of processing in that only one server is assumed and all fragments processed at the MU must be returned to the server prior to commit. All but the Semantics-based approach may violate durability. This is because local transactions which have committed may later be "undone" by a compensating transaction. It is certainly debatable as to whether this really violates durability, since the compensating transaction is a completely separate transaction. The request column indicates where the transaction is assumed to be requested. All but the Reporting assume it is requested at the Mobile Unit. Since this model is a more general than the others and not limited to a mobile computing environment, it does not assume that the initial request is made from any particular site. The Execute column indicates at what sites the kangaroo is assumed to execute. Again this really does not apply to the Reporting approach. The Kangaroo limits processing to nodes on the fixed network, while the Semantics approach assumes that the execution at a server on the fixed network is limited to the creation and then update of the fragments.

7.7 DATA CONSISTENCY IN INTERMITTENT CONNECTIVITY

Mobile clients encounter *wide variations* in connectivity ranging from high-bandwidth, low-latency communications through wired networks to total lack of connectivity [8, 15, 39]. Between these two extremes, connectivity is frequently provided by wireless networks characterized by low bandwidth, excessive latency, or high cost. To overcome availability and latency barriers and reduce cost and power consumption, mobile clients most often deliberately avoid use of the network and thus operate switching between connected and disconnected modes of operation. To support such behavior, *disconnected operation*–that is, the ability to operate in a disconnected mode–is essential for mobile clients [15, 16, 36, 47]. In addition to disconnected operation, an operation that exploits *weak connectivity*; that is, connectivity provided by intermittent, low-bandwidth, or expensive networks), is also desirable [14, 32]. Besides mobile computing, weak and intermittent connectivity also applies to computing using portable laptops. In this paradigm, clients operate disconnected most of the time, and occasionally connect through a wired telephone line or upon returning back to their working environment.

In the proposed scheme, data located at strongly connected sites are grouped together to form clusters. Mutual consistency is required for copies located at the same cluster, while degrees of inconsistency are tolerated for copies at different clusters. The interface offered by the database management system is enhanced with operations providing weaker consistency guarantees. Such weak operations allow access to locally (i.e., in a cluster) available data. Weak reads access bounded inconsistent copies and weak writes make conditional updates. The usual operations, called strict in this chapter, are also supported. They offer access to consistent data and perform permanent updates.

The scheme supports disconnected operation since users can operate even when disconnected by using only weak operations. In cases of weak connectivity, a balanced use of both weak and strict operations provides for better bandwidth utilization, latency, and cost. In cases of strong connectivity, using only strict operations makes the scheme reduce to the usual one-copy semantics. Additional support for adaptability is possible by tuning the degree of inconsistency among copies based on the networking conditions.

In a sense, weak operations offer a form of *application-aware adaptation* [33]. Application-aware adaptation characterizes the design space between two extreme ways of providing adaptability. At one extreme, adaptivity is entirely the responsibility of the application; that is, there is no system support or any standard way of providing adaptivity. At the other extreme, adaptivity is subsumed by the database management system. Since, in general, the system is not aware of the application semantics, it cannot provide a single adequate form of adaptation. Weak and strict operations lie in an intermediate point between these two extremes, serving as *middleware* between a database system and an application. They are tools offered by the database system to applications. The application can at its discretion use weak or strict transactions based on its semantics. The implementation, consistency con-

are similar to weak read-only transactions with no consistency requirements. ESR bounds inconsistency directly by bounding the number of updates. In Ref. [50] a generalization of ESR was proposed for high-level type specific operations on abstract data types. In contrast, our approach deals with low-level read and write operations.

In an *N-ignorant* system, a transaction need not see the results of at most N prior transactions that it would have seen if the execution had been serial [18]. Strict transactions are 0-ignorant and weak transactions are 0-ignorant of other weak transactions at the same cluster. Weak transactions are ignorant of strict and weak transactions at other clusters. The techniques of supporting N-ignorance can be incorporating in the proposed model to define d as the ignorance factor N of weak transactions.

7.13 CONCURRENCY CONTROL MECHANISM

Consistency-preserving execution is necessary for maintaining database consistency. In Chapter 5 a number of commonly known concurrency control mechanisms were discussed. This chapter investigates if any of them would work satisfactorily in mobile database systems.

Any scheme or mechanism, such as sorting, searching, concurrency control mechanism, system recovery, etc., has system overhead. In most cases a mechanism with least system overhead is preferred, even though it may not be efficient. This is especially true for mobile database systems where system overhead can create a serious performance problem because of low-capacity and limited resources. This is one of the main reasons for not considering conventional currency control mechanisms for serializing concurrent transactions for mobile database systems. However, they do provide a highly useful base for modified CCMs or for developing new ones. Some of these conventional CCMs can analyzed as follows:

7.13.1 Locking-Based CCMs

Two-phase incremental locking and simultaneous release is the most commonly used concurrency control mechanism. This scheme can be implemented on distributed database systems in three different ways: (a) centralized two-phase locking (primary site approach), (b) primary copy locking, and (c) distributed two-phase locking. It is useful to analyze if they are suitable for mobile database systems

Centralized Two-Phase Locking: In this scheme, one site (node) is responsible for managing all locking activities. Since the locking request traffic is likely to be very high, the central node should be almost always available. In a mobile database system, this requirement limits the choice of central node. A mobile unit cannot be a central node because (a) it is a kind of personal processing unit, (b) it is not powerful enough to manage locking requests, (c) it cannot maintain the status (locked or free) of data items, (d) it is not fully connected to other nodes in the network, and (e) its mobility is unpredictable. Base stations are the next choice, but they also have a number of

problems related mainly with functionality issues. A base station is a switch and is dedicated to providing services to mobile units. Adding transaction management functionality is likely to overload them, which would not be recommended by wireless service providers. Theoretically, this may be the best choice, and many researchers have selected base stations for incorporating database functions; however, in reality this is not an acceptable solution. A fixed host can be configured to act as a central node, but it is not equipped with a transceiver. As a result, it has to go through a base station to reach any mobile unit. No matter what component is identified as a central node, the problem of single-point failure cannot be avoided in this scheme.

Primary Copy Two-Phase Locking: This scheme eliminates a single point of failure and minimizes other problems of central node approach by distributing the locking responsibility among distributed to multiple sites. Each lock manager is now responsible for a subset of data items. The node executing a part of the transaction sends lock requests to appropriate lock manager. This approach does not solve the problem of identifying suitable sites for distributing locking responsibility. The choices are either base station or fixed hosts or both.

Distributed Two-Phase Locking: This scheme simply maximizes the extent of lock distribution. Here all nodes can serve as a lock manager. In the case of database partition this algorithm degenerates to centralized two-phase scheme. It is obvious that this scheme does not suggest a better selection of node for lock manager.

The other acceptable option for lock manager is to include separate database servers connected to base stations through wired network. One of the database servers can be identified as the central node for managing transactions under a centralized scheme, a subset of them for a primary copy scheme, and all for a distributed scheme. Out of all options, this seems to be a middle ground.

The communication overhead for managing locking and unlocking requests is another important problem to investigate. If a mobile unit makes a lock request on behalf of a transaction, it is executing and then (a) it will send the request to lock manager site (wireless message), (b) the lock manager will decide to grant or to refuse the lock and send the result to the mobile unit (wireless message), and (c) the mobile unit makes the decision to continue with forward processing or block or rollback depending upon lock manager's decision. Thus, each lock request will generate two wireless messages, which would become quite expensive with an increase in the workload. Furthermore, every rollback will generate an additional message overhead by restarting the transaction.

The amount of overhead closely related to the degree of consistency the database is programmed to maintain. To maintain stronger degree of consistency requires more resources compared to maintaining weaker degree of consistency. Thus one way of reducing the cost is to maintain weaker consistency level, and in many data processing situations a weaker consistency is acceptable. This is especially true for mobile database systems because mobile users are not likely to issue CPU-intensive large update transactions through their mobile units. If such a transaction is issued from a

laptop, then it could be executed at database servers with the strongest consistency level.

It would be hard to achieve maximum benefit only through a new CCM that maintains a weaker level of consistency. A new way of structuring and executing ACID transactions is also necessary. Very few CCMs for mobile database systems have been developed, and this section discuses a few of them.

Distributed HP-2PL CCM

In Ref. [28] a concurrency control mechanism called *Distributed HP-2PL (DHP-2PL)* is presented. This CCM is based on two phase locking and it is an extension of *HP-2PL* [1] CCM. It uses conflict resolution scheme of *Cautious Waiting* [19] mechanism to reduce the degree of transaction roll-backs.

In this scheme, each base station has a lock schedular which manages the locking requests for data items available locally. Each transaction, i.e., the holder of the data item (T_h) and the requestor of the data item (T_r) is assigned a unique priority. Thus when a requestor and a holder conflicts then their associated priority and their execution status (committing, blocked, etc.) are used to resolve the conflict. The steps are as follows.

- On a conflict check the priority of the holder and the requestor.
- If *Priority* (T_r) > *Priority* (T_h) *then check the status of* (T_h). *If* (T_h) *is not committing (i.e., still active* then check if it is a local transaction.
- If (T_h) is a local transaction then restart it locally. A local transaction accesses only those data items which are stored at the base station where the transaction originates.
- If (T_h) is a global transaction then restart it globally. A global transaction accesses data at more than one base stations. Roll-back of a global transaction requires communicating with all those base stations where the global transaction has performed some update operations.
- If (T_h) is in committing process then it is not restarted rather the (T_r) is forced to wait until (T_h) commits and releases all its lock. Adjust the priority of (T_h) as follows:
  Priority (T_h) := *Priority* (T_r) + *some fixed priority level.*
- if *Priority* (T_r) $\leq$ *Priority* (T_h) then block (T_r) until (T_h) commits and releases its locks.

A Cautious waiting approach is incorporated in the above method to minimize the impact of disconnection and unnecessary blocking. The modified algorithm is given below:

If *Priority* (T_r) > *Priority* (T_h) *and* T_h *is still active (not committing), then global or local restart* (T_h).
Else
If T_h *is a mobile client, then*
If the time T_h *spent at mobile unit > threshold, then ping the mobile unit. (The ping is done by the base station to check if* T_h *is active.)*
If there is no response, then restart T_h
Else
Block T_r. *This check is repeated at the end of a threshold.*
Endif
Else block T_r. *This checking is performed again when the time spent at the mobile unit is > threshold.*
Endif
Else Block T_r
Endif
Endif

The *threshold* is a function of average system performance which is used as a tuning parameter. This acts as a timeout value which helps to decide the status of a mobile unit. If the base station does not get a response from the mobile unit within the threshold value, then a disconnection is assumed. This may not be true but its effect is similar to a disconnection. The holder T_h is restarted even though it has a higher priority. This may increase the chances of missing the deadline for T_r.

Two more CCMs are discussed below. One takes the approach of weaker consistency, and the other uses transaction restructuring for developing CCMs for MDS.

7.13.2 CCM Based on *Epsilon* Serializability

A CCM based on *epsilon* serializability (ESR) [45] is presented here, which tolerates a limited amount of inconsistency. The mechanism is based on a two-tier replication scheme [12] that produces an epsilon serializable schedule. The scheme provides availability, accommodates the disconnection problem, and is scalable. It reduces transactions commit time and number of transaction rejections. ESR approach keeps the amount of inconsistency within a limit specified by $epsilon$. When $epsilon \rightarrow 0$, ESR reduces to conventional serializability situation. For example, in banking database a report that prints total summary in units of millions of dollars can tolerate inconsistency of a few hundreds dollars. Divergence control methods guarantee ESR the same way as concurrency control guarantee serializability. The concurrency control method that is presented here is a divergence control method to maintain ESR, which can be applied to a database whose state space is *metric*. Database state space depends on database semantics. Many practical applications with different semantics such as bank accounts, seats in airline reservation, and so on, are examples of metric state space. Bank database contains client names, addresses, account numbers, and account amounts but updates happen only to amount. Metric space S is defined as a state space having the following properties:

- A distance function *dist (u, v)* is defined over every $u, v \in S$ on real numbers; *dist (u, v)* is the difference between u and v, which represent database states.
- Triangular inequality, i.e., *dist (u, v) + dist (v, w) = dist (u, w).*
- Symmetry, i.e., *dist (u, v) = dist (v, u).*

In the mechanism, ESR [45, 52] is used to achieve acceptable reduction in consistency. ESR is an abstract framework, and an instance of ESR is defined by concrete specification of tolerated inconsistency. The CCM that is discussed here can also be applied on fragmentable, reorderable objects [57], which include aggregate item, such as sets, queues, and stacks.

The two-tier replication does not use traditional mechanisms (like two-phase locking or timestamping) and it provides availability and scalability, accommodates a disconnection problem, and achieves convergence. The basic idea of two-tier replication is first to allow users to run *tentative* transactions on mobile units, which makes tentative updates on the replicated data locally. When the mobile node connects to the database server, then these transactions are transformed to corresponding *base* transactions and re-executed at the servers. The base transactions are serialized on the master copy of the data and mobile units are informed about any failed base transactions. But the problem with this approach is that the mobile unit executes transaction without the knowledge of what other transactions are doing. This situation can lead to a large number of rejected transactions [3]. Another drawback is that transaction commit at MU tends to be large because these transactions know their outcome (i.e., committed or rejected) only after base transactions have been executed and the results are reported back to the MU. The CCM discussed here the two-tier replication scheme [12] is modified to reduce the number of rejected transactions and to reduce commit time of transactions executed at the MU. The BS can broadcast information to all the MUs in its cell.

A central server holds and manages the database $D = \{D_i\}$, where $i \in N$ is set of natural numbers and $D_i \in S$ where S is a metric space. Let d_i be the current value of the data object D_i. The data objects are replicated on the MU's and let n_i be the number of replicas of D_i in MDS. A limit Δ on the amount of change can occur on the replica at each MU, thus Δ_i denotes the change allowed in each replica of data object D_i on any MU. If the transaction changes the value of the data item by at most Δ_i in a MU, then they are free to commit; they do not have to wait for results of the execution of the base transaction on DBS. This reduces the commit time of the transactions and also the number of rejected transactions, which could happen due to the base transaction not being able to commit. To control the validity of Δ_i, a timeout parameter is defined whose value indicates a duration within which the value of Δ_i is valid. Timeout values of the data item should be some multiple I of broadcast cycle time T. The value I depends on the frequency of the incoming updates for the data item, and also it should be sufficiently large so that the MU's can send their updates within duration $I \times T$. The server will not update the value of the data item until time $I \times T$ has elapsed. It is assumed that the MUs take into consideration the uplink time and send their updates before the timeout expires at the server. The client

can disconnect from MDS during the timeout period and can perform updates. If the client disconnects for a period longer than timeout, then when it reconnects it should read the new values of Δ. If the updates are within the new limit set by Δ, then the MU can send the updates to the server; otherwise the MU will have to block some transactions so that total updates are within Δ. The blocked transactions will have to wait until the new values of Δ arrive at the MU. The steps of the algorithm go as follows:

At a DBS

1. Δ_i is calculated for each data object D_i. Δ_i is calculated using the function $\Delta_i = f_i\ (d_i, n_i)$. A function $f_i(d_i, n_i)$ is associated with each data object D_i, and it depends on the application semantics.

2. A timeout value τ is linked with Δ_i values of the data item.

3. DBS broadcasts the values of (d_i, Δ_i) for each data item and a timeout τ for these values at the beginning of the broadcast cycle.

4. The DBS either receives pre-committed transactions (transactions which have made updates to the replicas on the MU and committed) or can receive request transactions (transactions which are directly sent to the DBS by the MU). A transaction that violates the limit is not executed at an MU, because it could change the value of replica D_i by more than Δ_i at the MU. It is sent to the DBS as request transaction for execution on the master database.

5. The DBS serializes the pre-committed transactions according to their order of arrival. After the timeout expires, the DBS executes a request transaction, reports to the MU whether the transaction was committed or aborted, and repeats the procedure from the first step.

At MU

- MU has the value of (d_i, Δ_i) and timeout τ for every data item D_i it has cached.

- MU executes transaction t_i. It changes the current value of D_i by Δ_{i-t_i}. Let $\Delta_{i-c}i$ be the current value of the total change in D_i since its last broadcast of value Δ_i).

- The value Δ_{i-t_i} is added Δ_{i-c}. The following cases are possible depending on the value of Δ_{i-t_i} and Δ_{i-c}:

    1. If $\Delta_{i-t_i} \leq \Delta_i$ and $\Delta_{i-c} \leq \Delta_i$, then t_i is committed at MU and it is sent to DBS for re-execution as a base transaction on the master copy.
    2. If $\Delta_{i-t_i} \leq \Delta_i$ and $\Delta_{i-c} > \Delta_i$, then t_i is blocked at MU until new set of (D_i, Δ_i) is broadcasted by the server.

3. If $\Delta_{i-t_i} > \Delta_i$ then t_i is blocked at MU and submitted to the server as a request transaction.

7.13.3 Relationship with ESR

The mechanism for maintaining ESR has two methods: (a) *divergence control (DC)* and (b) *consistency restoration*. This section discuses these methods and show their use in developing the concurrency control mechanism.

A transaction imports inconsistency by reading uncommitted data of other transactions. A transaction exports inconsistency by allowing other transaction to read its uncommitted data. Transactions have *import* and *export* counters. The following example shows how these counters are maintained.

$t_1 : w_1(x),\ w_1(z);$
$t_2 : r_2(x),\ r_2(y);$
$t_3 : w3(y),\ r3(z).$

Schedule: $w_1(x), r_2(x), r_2(y), w_3(y), r_3(z), w_1(z)$

In the above execution, t_2 reads from t_1. So it is counted as t_1 exporting one conflict to t_2 and t_2 importing one conflict from t_1. So an export counter of t_1 is incremented by 1, and an import counter of t_2 is incremented by 1. Transaction t_3 does not import or export any conflicts. The divergence control (DC) method sets limit on conflicts by using import and export limits for each transaction. Thus, update transactions have export limit and query transactions (read-only) have import limit, which specify the maximum number of conflicts they can be involved in. When import *limit* > *0* and export *limit* > *0*, then successive transactions may introduce unbounded inconsistency. For example, t_1 may change the value of a data item by a large amount and t_2 will read this value and operate on it as import and export counters are not violated. Later if t_1 aborts, t_2 would have operated on a value that was deviated from consistent value by a large amount. This situation requires consistency restoration, which is done by consistency restoration algorithms.

In this concurrency control mechanism, DC sets limits on the change allowed in each data item value at MU and does not allow transactions to violate this limit. If it does, then it is sent as a request transaction to DBS for execution. In this scheme a transaction at MU will see an inconsistent value of data item for a maximum period of τ (the timeout period) after which it receives new consistent values of the data items. During τ, the value of data item d_i may diverge from the consistent value by a maximum of $N_i \times \Delta_i$, where N_i is the number of replicas of d_i. In this way transactions are allowed to execute on inconsistent data item but the inconsistency in data value is bounded by $N_i \times \Delta_i$. So in this CCM the DC includes the function $fi(d_i, n_i)$, which calculates Δ_i for each d_i and also for the algorithm executed at MU to execute transactions. Thus, the consistency restoration includes the execution of request and pre-committed transactions at DBS and broadcasting of the consistent value of the data item to the MU.

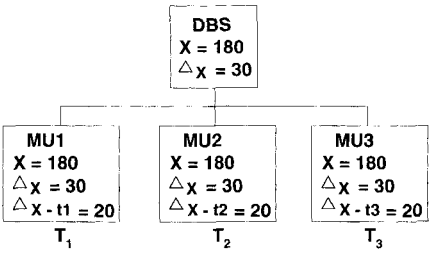

Fig. 7.12 Execution of transactions under this CCM.

■ EXAMPLE 7.7

This example explains the working of the CCM discussed in this section. Figure 7.12 illustrates the execution of concurrent transaction under this CCM. Suppose a data item X represents total number of movie tickets. X belongs to metric state space. Let N_x be the number of replicas of X. Initially suppose $X = 180$ and $N_x =$ and X is replicated at MU1, MU2 and MU3. The functions $f_x(X, N_x)$ that calculates Δ_x is $\Delta_x = f_x(X, N_x) = (X/2)/N_x = X/2N_x = 30$. Here X is divided by 2 to keep some tickets for the request transaction, which cannot be executed at the MU. This function depends on the application semantics and the policy the application developer wants to follow. Each data item will have different function depending on the semantics of that data item. (Δ_x, X, τ), where τ is timeout within which the MU should send committed transaction for re-execution at the server, is broadcasted by the DBS server to MU's. The following three cases arise:

Case 1: Transactions t_1, t_2, and t_3 arrive at MU1, MU2, and MU3, respectively. Consider the case where t_1 books 20 tickets, t_2 books 30 tickets, and t_3 books 40 tickets. Figure 7.13 shows the state of the system at this instant. Suppose Δ_x represents change in value of data item X. Each MU that has a replica of X will maintain the value Δ_{x-c}.

At MU1: Initially $\Delta_{x-c} = 0$.

t_1 books 20 tickets, so $\Delta_{x-t_1} = 20$ and $\Delta_{x-c} = \Delta_{x-c} + \Delta_{x-t_1} = 20$. As $\Delta_{x-c} < \Delta_x$, t_1 is committed at MU1 and so X is updated to 160 and t_1 is sent to DBS for re-execution on the master copy.

At MU2: Initially $\Delta_{x-c} = 0$

t_2 books 30 tickets, so $\Delta_{x-t_2} = 30$ and $\Delta_{x-c} = \Delta_{x-c} + \Delta_{x-t_2} = 30$. As $\Delta_{x-c} < \Delta_x$, t_2 is committed at MU2 and X is updated to 150 and t_2 is sent to DBS for re-execution.

At MU3: Initially $\Delta_{x-c} = 0$.

t_3 books 40 tickets and makes $\Delta_{x-t_3} = 30$ and $\Delta_{x-c} = \Delta_{x-c} + \Delta_{x-t_3} = 40$. Since $\Delta_{x-c} > \Delta_x$, t_3 is not executed at MU3 and is sent as request transaction to DBS for execution.

DBS receives t_3, t_2, and t_1 in this order. Since t_3 is a request transaction, it is executed after timeout τ has expired and after the execution of t_2 and t_1 on the master copy. So the execution at DBS is $X = 180$, t_2, $X = 150$, t_1, $X = 130$, t_3, $X = 90$; and after the execution, Δ is recomputed using the function $f_x(X, N_x)$. Thus, $\Delta_x = f_x(X, N_x) = X/2N_x = 15$. The DBS broadcasts ($X = 90$, $\Delta_x = 15$, τ) and each MU now can update the value of X by not more that 15 and sends the transaction for re-execution within τ. Figure 7.13 illustrates case 1.

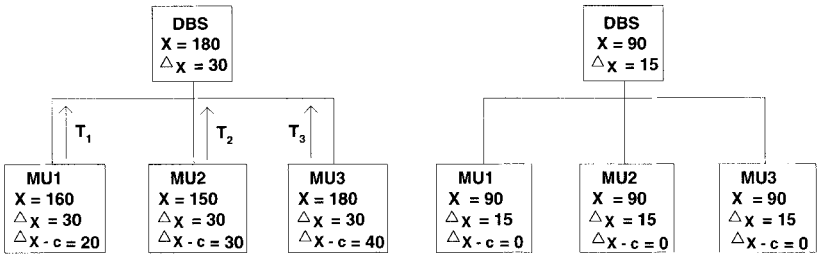

Fig. 7.13 Intermediate state in CCM.

Transactions on MU see an inconsistent value of the number of tickets only for period τ; after that, DBS sends their consistent value. The transactions that want to know number of tickets available will get an approximate value of number of tickets. Inconsistency in the value of data objects is bounded by refreshing the data object value at a regular interval of τ and setting a limit on Δ on the maximum update that can be made during that period.

Case 2: MU3 receives the values ($X = 90$, $\Delta_x = 15$, τ) from the DBS. For every new timeout value, Δ_{x-c} is reset to zero. Transactions t_4 and t_5 arrive at MU3. t_4 books 10 tickets and t_5 books 8 tickets. Suppose the execution order is t_4, t_5. After the execution of t_4, $\Delta_{x-t_4} = 10$ and $\Delta_{x-c} = \Delta_{x-c} + \Delta_{x-t4} = 10$. As $\Delta_{x-c} < \Delta_x$, t_4 is executed at MU3 and is sent to the DBS for re-execution. Before τ expires, transaction t_5 arrives at MU3 where $\Delta_{x-t_5} = 10$ and $\Delta_{x-c} = 8$. This makes $\Delta_{x-c} = \Delta_{x-c} + \Delta_{x-t_5} = 18$. As $\Delta_{x-c} > \Delta_x$, t_5 is not executed at MU3 and sent to DBS for execution as a request transaction.

Case 3: MUs receive values ($X = 80$, $\Delta_x = 0$, τ) from DBS. t_6 arrives at MU2 and books 2 tickets. At MU2 initially, $\Delta_{x-c} = 0$. t_6 books 2 tickets and makes $\Delta_{x-t_6} = 2$. At this point, $\Delta_{x-c} = \Delta_{x-c} + \Delta_{x-t_6} = 2$. As $\Delta_{x-c} > \Delta_x$, t_6 is not executed at MU2 but is sent as a request transaction to the DBS.

All update transactions that arrive at MU will be sent to DBS as request transaction because no change is allowed to the replica at MU since $\Delta_x = 0$. Only read transactions at MU can access data item X.

7.14 TRANSACTION COMMIT

The distributed execution of a transaction requires collaboration among nodes to commit the transaction. The collaboration is initiated and managed by the coordinator, which makes sure that every subtransaction is executed successfully and ready to commit. If any of its subtransactions cannot commit, then the parent transaction is aborted by the coordinator.

The entire process of commit has two phases: (a) checking the intention of each node participating in the execution of a transaction (participants) and (b) collecting the intensions of participants and committing the transaction. The entire process is atomic, and the commit protocol is referred to as *Atomic Commitment Protocol (ACP).*

The most common ACP used in conventional distributed database systems is called a *Two-Phase Commit* (2PC) protocol. There is a *Three-Phase Commit* (3PC) protocol [4] which claims to be more efficient than 2PC but requires a higher number of messages compared to 2PC for making a commit decision. So far, no system has implemented 3PC, but it continues to be an interesting research topic.

7.14.1 Two-Phase Commit Protocol – Centralized 2PC

A distributed database system with multiple nodes is assumed to describe 2PC. A transaction T_i originates at a node which assumes the role of coordinator for T_i. The coordinator fragments T_i and distributes them to a set of participants. The coordinator may or may not keep a fragment for itself. Thus a coordinator and a set of participants together executes T_i leading either to a commit or to an abort as decided by the coordinator. The protocol makes sure of the following:

- **Participants' decision:** All participants reach a decision for their fragments. All decision can be either Yes or No.
- **Decision change:** A participant cannot change its final decision.
- **Coordinator's decision:** The coordinator can decide to commit T_i only if all participants and the coordinator agree to commit their subtransactions. It is not that in this situation the coordinator has no other option than to decide commit; it can still abort the transaction.

  When the failure scenario is included, then the following additional steps are required for making some decision.

- **No failure:** In the absence of any failure, if all processing nodes (participants and coordinator) agree to commit, then the coordinator will commit T_i.

- **With failure:** Failure of one or more participants or the coordinator may delay the decision. However, if all failures are repaired within *acceptable* time, then the coordinator will reach a decision. This identifies the *non-blocking* property of centralized 2PC. The non-blocking property is essential for any APC. However, it may not be strictly enforced; that is, a failure may generate infinite blocking situation.

The working of centralized 2PC is described in the following steps [4]:

1. **Transaction fragmentation and distribution:** A transaction T_i arrives to a node. This node servers as the coordinator for T_I. The coordinator fragments T_i into subtransactions and distributes these fragments to a set of participants. These nodes begin executing their subtransactions of T_i.

   First phase of centralized 2PC – Voting phase

2. **Voting:** The coordinator multicasts a message (vote request – VR) to all participants, asking them to vote if they can commit their subtransaction of T_i.

3. **Participants' vote:** When a participant receives VR message from the coordinator, it composes its response (vote) and sends it to the coordinator. This response can be a Yes or a No. If the vote is Yes, then the participant enters into an "uncertainty" period after sending it to the coordinator . During this period a participant cannot proceed further (make any unilateral decision in behalf of its subtransaction of T_i) and just waits for an abort or commit message from the coordinator. If the vote of the participant is No, then it does not wait for coordinator's response and aborts its subtransaction and stops.

   Second phase of centralized 2PC – Decision phase

4. **Commit decision and dispatch:** When the coordinator receives votes from all participants and has its own vote, it performs an AND operation among these votes. If the result is a YES, then the coordinator decides to commit otherwise it decides to abort T_i. It multicasts the decision to all participants and stops.

5. **Participants decision:** All participants receives a coordinator's decision and act accordingly. If the decision is to abort, all participants abort their fragments and then stop.

7.14.2 Node Failure and Timeout Action

In order to make sure that the non-blocking property of centralized 2PC is effectively implemented, the occurrences of infinite wait because of node failure must be dealt with. One of the schemes to enforce non-blocking property is to use timeout action. A timeout value identifies how long a participant should wait for the anticipated message before its takes some action.

In the description of centralized 2PC participants wait for VR messages from the coordinator at the beginning of step 3, and the coordinator waits for participants' Yes

or No decision at the end of step 3. Similarly, all participants either wait for the coordinator's commit or abort message in step 5. If a participant times out at the beginning of step 3, then it can unilaterally decide to abort because it is not in its uncertainty period. At the end of step 3, the coordinator may time out waiting for Yes or No messages from some or all participants. It may decide to abort and send abort messages to those participants who did not send their vote and who send Yes votes.

The timing out of participants at the beginning of step 4 is more involved because participants are in their uncertainty period and they cannot change their vote from Yes to No and abort their subtransactions. It is possible that the coordinator might have sent the commit message and it reached only to a subset of participants. If a participant times out in its uncertainty period and decides to abort, then it would be a wrong action. To take care of this immature abort by a timed-out participant, a *cooperative termination protocol* can be used. A cooperative termination protocol helps a participant to gather information about the last message from the coordinator which this participant missed and timed out.

Cooperative Termination Protocol: When a participant in uncertainty period fails to receive a commit or abort message from the coordinator, then it has two options: (a) Ask the coordinator about its last message or (b) Ask one of is neighbor participants. In case (a) if the coordinator is available to respond, then it can get the desired information and decide accordingly. To use (b), every participant must know the identity of all other participants. This can be easily provided by the coordinator at the time of sending an VR message in step 2. The following three cases arise when (b) is used:

1. Participant P1 asks P2 about the final outcome. If P2 has decided to commit or abort (it did receive coordinator's decision message), then it can inform P1 about its decision and P1 can act accordingly.

2. P2 is not in uncertainty period and has not voted yet. It decided to abort and informs P1. P1 also aborts.

3. P2 has voted Yes but has not received a decision from the coordinator either. P2 is in a similar situation, that is, it timed out in its uncertainty period as P1 and cannot help. P1 and P2 can continue to ask other participants, and hopefully at least one participant might have received the coordinator's decision. It informs P1, and P1 acts accordingly and so does P2. If all participants are timed out and did not get coordinator's decision, then possibly the coordinator has failed and they can abort and stop.

The performance of a commit protocol largely depends on the number of messages it uses to terminate (abort of commit) a transaction. To evaluate the cost of communication, two parameters are used: (a) time complexity and (b) message complexity.

Time Complexity: This parameter evaluates the time to reach a decision. It includes the time to complete a number of other necessary activities such as logging messages,

preparing messages, etc. A smaller value is highly desirable. The decision time in the absence of any kind of failure (coordinator or participants of both) is obviously smaller compared to the time with failure. In the absence of failures, the protocol uses three message *rounds*. A message round is the total time the message takes to reach from its source to the destination. The first message round is the broadcast of VR messages to the participants from the coordinator; in the second round, all participants send their votes; and in the third round the coordinator broadcasts its decision (commit or abort). In the presence of failures, two additional rounds are required: (a) a timed-out participant enquires the coordinator's decision and (b) a response from a participant who received coordinator's decision (this participant is out of its uncertainty period). Thus with no failure of any kind, three message rounds–and with failure, five message rounds–are required to terminate a transaction.

Message Complexity: Message complexity evaluates the number of message exchanged between destinations and sources to reach a decision. In a centralized 2PC, message exchange takes place between one coordinator and n participants when there is no failure. The total number of messages exchanged is $3n$ in the three steps of the protocol:

- The coordinator sends VR message to n participants = n messages.
- Each participant sends one vote message (Yes or No) to coordinator = n vote messages.
- The coordinator sends decision message to n participants = n messages.

  In the presence of failure, each timed-out participant who voted Yes initiates cooperative termination protocol. In the worst-case scenario, all participated could be timed out and initiate the protocol. If there are m such timed-out participants with Yes vote, then $m \leq n$.
- m participants will initiate the protocol and send $n - 1$ decision request messages. The requestor participant m_i will get a response from at least one of the n participants and would come out of its uncertainty period. This cycle will continue until m participants come out from their uncertainty period. The total number of messages used in this entire process will be

$$m(n-1) + \sum_{i=1}^{m}(n - m + i) \quad = \quad 2m(n-1) - m^2/2 + m/2$$

  To minimize the time or message complexity or both, two variations of 2PC exist: (a) *decentralized 2PC* and (b) *linear or nested 2PC*.

7.14.3 Decentralized 2PC

In this scheme the coordinator minimizes the message complexity by sending its vote along with VR message to participants. If the coordinator's vote is No, then participants know the decision and abort their subtransactions and stop. If the coordinator's

vote is Yes, then each participant sends its vote to all other participants. After receiving all votes, each participants decides. If all votes are Yes, then the transaction commits; otherwise it is aborted.

Time Complexity: In a decentralized 2PC there are two message rounds: (a) The coordinator sends a VR message and its vote, and (b) the participant sends its Yes vote and transaction commits. When the coordinator sends its Yes vote to participants, it implies that "I am ready to commit and if you are also, then go ahead and commit" and, therefore, there is no need for the coordinator to send a Commit vote. This reduces one message round compared to centralized 2PC.

Message complexity: The reduction in time complexity unfortunately increases the message complexity. In a centralized 2PC the coordinator makes the final decision but in distributed 2PC everybody participates in the decision process. This requires that each participant communicates with all other participants to know their votes. If there are n participants, this process requires n^2 messages. Thus the total number of messages to commit a transaction in failure as well as in no failure cases is n^2 (participant to $n-1$ participants) + n (coordinator to n participants). In the case of $n > 2$, decentralized 2PC always takes a greater number of messages than a centralized 2PC.

7.14.4 Linear or Nested 2PC

In linear 2PC the message complexity is reduced by collecting votes serially. All participants and the coordinator are ordered linearly. Each participant has a left and a right neighbor and a coordinator has only one neighbor. Figure 7.14 illustrates the setup.

Fig. 7.14 Linear ordering of participants and coordinator.

The protocol works as follows:

1. The coordinator sends its Yes or No vote to participant P_1.
2. P_1 performs *Coordinator's vote* $\bigwedge$ P_1's *vote* = X. X = Yes or No.
3. P_1 sends X to P_2 and the process continues until the result reaches to P_n.
4. If the outcome of P_n computation is Yes, it decides to commit and sends this message to P_{n-1}.
5. The return message containing the commit decision finally reaches to the coordinator and completes the commit process.

Time Complexity: There is no message broadcast in a linear 2PC, so it requires the same number of rounds as the number of messages to make the final decision. Thus with n participants it will require $2n$ rounds, which is much larger than centralized and decentralized 2PC.

Message Complexity: With n participants, this protocol requires $2n$ messages: n forward messages and n return messages which is much smaller than the message complexity of decentralized and centralized 2PC.

Table 7.6 compares the message and time complexity of centralized, decentralized, and linear 2PC with no failure. It is hard to identify the most efficient protocol for all systems because of the wide ranging values of parameters such as message size, communication speed, processing delay, etc., which are highly system-dependent. However, it can be seen that the centralized 2PC offers a good compromise.

Table 7.6 Message and time complexity in various 2PC

Protocols	Messages	Rounds
Centralized	$3n$	3
Distributed	$n^2 + n$	2
Linear	$2n$	$2n$

7.15 COMMITMENT OF MOBILE TRANSACTIONS

7.15.1 Commit Protocols for Mobilaction

The mobility and other characteristics of *MUs* affect Mobilaction processing especially its commitment. Some of the common limitations are: (a) An *MU* may cease to communicate with its *BS* for a variety of reasons, (b) it may run out of its limited battery power, (c) it may run out of its disk space, (d) it may be affected by airport security, (e) physical abuse and accident, (f) limited wireless channels for communication, and (g) unpredictable *handoffs*.

A mobile computing environment creates a complex distributed processing environment; therefore, it requires a distributed commit protocol. We have assumed the two-phase commit approach as the basis of developing our mobile commit protocol. One of the essential requirements of distributed processing is that all subtransactions of T_i must be ready to commit. In MDS a complete knowledge of this state becomes relatively more complex because of mobility. It is crucial that the scheme to acquire this knowledge must use minimum message communication, and it is also important that this scheme should not be dependent on the mobility of the involved MUs.

The different types of data (temporal and spatial) in mobile computing provide more freedom in designing commit protocols. Like conventional distributed database systems, a transaction in MDS may be processed by a number of *DBSs* and *MUs*; therefore, some commit protocol is necessary for their termination. Legacy commit

protocols such as *2PC (two-phase commit), 3PC (three-phase commit)* [4], etc., will not perform satisfactorily mainly because of limited resources, especially wireless channel availability. For example, the most commonly used *2PC* uses three message rounds in the case of no failure and uses five in the case of failure for termination [4]. Note that it requires additional support (use of timeout) for termination in the presence of blocked or failed subtransactions. Thus, the time and message complexities are too high for MDS to handle and must be minimized to improve the utilization of scares resources.

The mobility of *MU* adds another dimension to these complexities. It may force MDS to reconfigure the initial commit setup during the life of a transaction [22]. For example, a proper coordination among the subtransactions of a transaction under a *participants-coordinator* paradigm may be difficult to achieve with the available resources for its commitment. Mobile database systems, therefore, require commitment protocols which should use a minimum number of wireless messages, and *MU* and *DBSs* involved in T_i processing should have independent decision-making capability and the protocol should be *non-blocking* [4].

7.16 TRANSACTION COMMITMENT IN MOBILE DATABASE SYSTEMS

The mobility and other characteristics of MUs affect transaction processing, especially its commitment. Some of the common limitations are: (a) An MU may cease to communicate with its BS for a variety of reasons, (b) it may run out of its limited battery power, (c) it may run out of its disk space, (d) it may be affected by airport security, (e) physical abuse and accident, (f) it has limited wireless channels for communication, and (g) unpredictable *handoff*.

Like conventional distributed database systems, a transaction in MDS may be processed by a number of nodes such as DBSs and MUs; therefore, some commit protocol is necessary for their termination. Conventional commit protocols such as 2PC, 3PC [4], etc., could be molded to work with MDS; however, they will not perform satisfactorily mainly because their resource requirements may not be satisfied by MDS on time. For example, the most commonly used centralized 2PC uses three message rounds in the case of no failure and uses five in the case of failure for termination [4]. It requires additional support (use of timeout) for termination in the presence of blocked or failed "subtransactions." Thus, the time and message complexities are too high for MDS to handle and must be minimized to improve the utilization of scares resources (wireless channel, battery power, etc.)

The mobility of *MU* adds another dimension to these complexities. It may force MDS to reconfigure the initial commit setup during the life of a transaction [21, 22, 23]. For example, a proper coordination among the subtransctions of a transaction under *participants-coordinator* paradigm may be difficult to achieve with the available resources for its commitment. For example, a mobile unit may not receive coordinator's vote request and commit messages and it may not send its vote on time because of its random movement while processing a subtranaction. This may generate unnecessary transaction aborts. These limitations suggests that MDS commit protocol must

support independent decision-making capability for coordinator and for participants to minimize cost of messages. A new commit protocol is required for MDS which should have the following desirable properties:

- It should use a minimum number of wireless messages.
- *MU* and *DBSs* involved in T_i processing should have independent decision-making capability, and the protocol should be *non-blocking*.

An analysis of conventional commit protocols indicates that timeout parameter could be used to develop a commit protocol for MDS. In conventional protocols, timeout parameter is used to enforce non-blocking property. A timeout identifies the maximum time a node can wait before taking any decision. The expiration of timeout is always related to the occurrence of some kind of failure. For example, in conventional 2PC the expiration of timeout indicates a node failure and it allows a participant to take a unilateral decision.

If a timeout parameter can identify a failure situation, then it can also be used to identify a success situation. Under this approach the end of timeout will indicate a success. The basic idea then is to define a timeout for the completion of an action and assume that at the end of this timeout the action will be completed successfully. For example, a participant defines a timeout within which it completes the execution of its subtransaction and sends its update through the coordinator to DBSs for installing it in the database. If the updates does not arrive within timeout, then it would indicate a failure scenario. The coordinator does not have to query the participant to learn about its status.

Recently timeout parameter has been used in a nonconventional way for developing solutions to some of the mobile database problems. This section presents a commit protocol which is referred to as *Transaction Commit on Timeout (TCOT)* [21, 22]. It uses timeout parameter to indicate a success rather than a failure.

The TCOT protocol is discussed below in detail. A transaction T_i is fragmented into several subtransactions, which are distributed for execution among a number of DBSs and the MU where T_i originated. These nodes are defined as *Commit Set* of T_i; the MU where T_i originates is referred to as *Home MU* (MU_H); and the BS of MU_H is referred to as Home BS (BS_H).

Definition 7.21 *A* **commit set** *of a T_i is defined as the set of DBS and the MU_H, which take part in the processing and commit of T_i. A DBS is identified as a static member, and the MU is a mobile member of a commit set.*

TCOT strives to limit the number of messages (especially uplink). It does so by assuming that all members of a commit set successfully commit their fragments within the timeout they define after analyzing their subtransactions leading to commit of T_i. Unlike 2PC or 3PC, no further communications between the CO and participants take place for keeping track of the progress of fragments. However, the failure situation is immediately communicated to CO to make a final decision.

It is well known that finding the most appropriate value of a timeout is not always easy because it depends on a number of system variables, which could be difficult to

quantify. However, it is usually possible to define a value for timeout, which performs well in all cases. An imprecise value of timeout does not affect the correctness but affects the performance of the algorithm.

Every CO (new or existing) must know the identity of each member of a commit set. Every MU_H stores the identity of its current CO for each transaction requested there. When MU_H moves to another cell, then during registration it also informs the BS about its previous CO. As soon as MU_H sends T_i to BS_H, the latter assumes the role of CO for T_i. In the dynamic approach also the transfer of CO does not require extra uplink or downlink messages because the notification process is a part of the registration.

Types of Timeout

TCOT protocol uses two types of timeout: *Execution Timeout* (E_t) and *Update Shipping Timeout* (S_t).

Execution Timeout (E_t): This timeout defines a value within which a node of a commit set completes the execution (not commit) of its execution fragment or subtransaction e_i. It is an upper bound of the time a DBS or the MU_H requires to complete the execution of e_i.

The CO assumes that the MU_H or a DBS will complete the execution of its e_i within E_t. The value of E_t may be node-specific. It may depend on the size of e_i and the characteristics of the processing unit; thus, $E_t(MU_i)$ may or may not be equal to $E_t(MU_j)$, $(i \neq j)$. We identify MU_H's timeout by $E_t(MU)$ and identify DBS's timeout by $E_t(DBS)$. The relationship between these two timeouts is $E_t(MU)$ = $E_t(DBS)$ $\pm\Delta$. The Δ accounts for the characteristics such as poor resources, disconnected state, availability of wireless channel, etc., compared to DBS. It is possible that a MU may take less time than its E_t to execute its e_i. We also do not rule out the possibility that in some cases E_t(DBS) may be larger than $E_t(MU_H)$. E_t typically should be just long enough to allow a fragment to successfully finish its entire execution in a normal environment (i.e., no failure of any kind, no message delay, etc.)

Shipping timeout (S_t): This timeout defines the upper bound of the data shipping time from MU_H to DBS.

In E_t, the cached copy of the data is updated at the MU. To maintain global consistency, all data updates done by the MU_H must be shipped and installed at the database located at DBS. Thus, at the end of E_t the CO expects the updates to be shipped to the DBS and logged there within S_t.

7.16.1 TCOT Steps–No Failure

In TCOT three components, MU_H, CO, and DBSs, participate. The steps in the absence of any kind of failure are:

- *Activities of MU_H*:
  - A T_i originates at MU_H. The BS_H is identified as the CO. MU_H extracts its e_i from T_i, computes its E_t, and sends $T_i - e_i$ to the CO along with the E_t of e_i. MU_H begins the processing of e_i.
  - While processing e_i, MU_H updates its cache copy of the database, composes update shipment, and appends it to the log.
  - During processing, if it is determined that e_i will execute longer than E_t, then MU_H extends its value and sends it to CO. Note that this uses one extra uplink message. The frequency of such extension requests can be minimized with a careful calculation.
  - If the local fragment e_i aborts for any reason, then MU_H sends an Abort message to CO (failure notification).
  - After execution of e_i MU_H sends log of updates to the CO. The updates must reach to CO before S_t expires. It could be possible that updates may reach CO much earlier, in which case it may decide commit sooner.
  - In the case of read-only e_i, MU_H sends a commit message to CO. This is not an extra message, it just replaces shipping update message.
  - Once the updates are dispatched to CO, MU_H declares commit of e_i. Note that the underlying concurrency control may decide to release all the data items to other fragments. If for some reason T_i is aborted, then fragment compensation may be necessary.
  - If MU_H fails to send updates to CO within S_t and it did not extend E_t, then the CO aborts e_i.
- *Activities of CO*:
  - Upon receipt of $T_i - e_i$ from MU_H, the CO creates a *token list* for T_i, which contains one entry for each of its fragments. Figure 7.15 shows a token list entry for e_i of T_i. In the case of CO change, a token is used to inform the new CO the status of fragment and commit set members. The CO splits $T_i - e_i$ into e_j's ($i \neq j$) and sends them to the set of relevant DBSs.

e_i	E_t (MU_i) or E_t (DBS_j)	Coordinator ID	Commit set

Fig. 7.15 An entry of a token list.

  - After receiving E_t from a DBS, the CO constructs a token for that fragment and keeps it for future use.
  - If a new E_t (extension) is received either from MU_H or from a DBS, then the CO updates the token entry for that fragment.

- CO logs the updates from MU_H.
- If the CO has MU_H's shipment before S_t expires and commit messages from other DBSs of the commit set, then the CO commits T_i. At this time the updates from the MU_H are sent to the DBSs for update to the primary copy of the databases. Note that no further message is sent to any member of the commit set of T_i.
- If CO does not receive updates from MU_H within the timeout or does not receive commit message from any of DBSs of the commit set, then it aborts T_i and sends a Global Abort message (wired message to DBSs and wireless to MU) to those members of the commit set who committed their fragments.

• *Activities of DBS*:

- Each *DBS*, upon receiving its fragment, computes E_t and sends it to the CO. *DBS* begins processing its fragment and updates its own database.
- If it is determined that the fragment will execute longer than E_t, then this value is extended and the new value is sent to the CO.
- At the end of e_j, it sends a "commit message" to the CO.
- If *DBS* cannot complete the execution of its e_j for any reason and did not extend E_t, then it sends an Abort message to the CO.

Discussion: One may argue that either E_t or the "commit message" is sufficient for making a commit decision. This is not entirely correct. E_t identifies when a fragment will finish its execution and will be ready to commit. Thus, at the end of E_t, CO will assume that the *DBS* has committed its fragment, which may not be true (fragment may not have been processed because of the failure of the *DBS*). Since a *DBS* does not ship updates, it must use a message for informing the status of the fragment. On the other hand, if there is only "commit message," then the CO could never get this message from a *DBS* for some reason and wait for ever to make the final commit decision. Thus, for making the final decision and doing it efficiently, both E_t and "commit message" are necessary. Note that a *DBS* communicates with the CO through wired channel and any extra message does not create any message overhead.

TCOT, unlike 2PC, has only one phase commit operation. No vote-request or commit message is sent to commit set members. The task assignment message to these members provides necessary information and directives for completing commit. Only in the case of abort, one extra wireless message is used. In reality, not many transactions are aborted and this extra message not likely to generate noticeable overhead.

In the case of a read-only fragment, MU_H does not send any update to the CO; but similar to a DBS, it sends only a "commit" message.

7.16.2 Node Failure–Fragment Compensation

The process of compensation is not related to commit; rather, it comes under recovery [17] but it becomes an issue for long running transaction. In TCOT a member of T_i's commit set may commit its fragment, and the underlying concurrency control (two-phase locking scheme is assumed) may decide to release its data items to other concurrent fragments before the CO declares the commit of T_i. For example, if $e_i(T_i)$ is committed by MU_H but T_i is aborted, then e_i must be compensated. When MU_H receives a message to abort e_i from the CO, then, if possible [17], a compensating transaction for e_i is executed. At the end of compensation, MU_H informs CO and sends new updates if there is any. Figure 7.16 illustrates the relationship among E_t, S_t, abort, and compensation. After S_t the MU_H can make data items available to e_j $(i \neq j)$. This means that after S_t an e_i may be compensated.

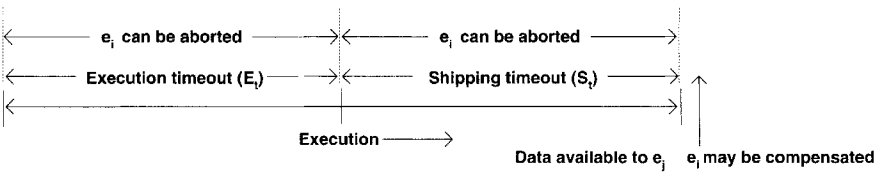

Fig. 7.16 Relationship between E_t and S_t, abort, and compensation.

7.16.3 TCOT with Handoff

Updates from MU_H and dispatch of commit message from DBSs in the case of a handoff must be sent to the right CO if it changes. The change in CO is notified using a *token*. The following steps define the commit process in the presence of a handoff.

- MU_H moves to a different cell and it registers with the new BS.
- If MDS employs dynamic selection of CO, then the MU_H sends the identity of its last CO in the registration process and accepts the new BS as its next CO. The new BS gets the *token* from the last CO, which provides necessary information.
- The new CO identifies other members of the commit set from the *token* and notifies them about the change of CO. Note that the communication between the new CO and DBSs is through wired channel. The processing of T_i resumes normally.

A doze mode of MU_H will mainly affect its E_t. MU_H may not be aware of its movement, but it knows when it enters into the doze mode. Therefore, before entering doze mode, MU_H can always request for extension to its E_t. If granted, then the fragment will not be aborted; otherwise a global abort will be initiated.

7.16.4 Special Cases

A number of special cases may arise during commit, and TCOT manages them as follows:

- S_t **expires before DBSs sends commit message:** It is possible that MU_H commits its e_i and sends its updates to the CO, before DBSs send their commit messages to the CO. In this case the CO will wait for the commit messages.
- **DBSs send commit messages before** S_t **expires:** The CO will wait for S_t to expire before making any decision.
- S_t **expires but no updates or no commit message:** The CO will send abort message to the members of the commit set.

Note that the abort could be received at any time. If it is received prior to commit, then a local abort with corresponding undo is needed. If, however, it is received after the local commit, then a compensation is needed. Further, when a fragment is executed, the decision to commit or abort is made locally. However, the implicit assumption is that a global commit occurs.

7.16.5 An Alternate TCOT Protocol

In the first version of TCOT, MU_H is responsible for extracting e_i from T_i, computing E_t, and sending $T_i - e_i$ to the CO. In this approach, every T_i is examined by MU_H, which is not necessary. This can be improved by sending the entire T_i to the CO and letting the CO do the fragmentation, estimate E_t, and send the information back to MU_H. This will use one extra wireless downlink message but reduces the workload of MU_H since many T_i's may not be processed by MU_H. The other advantage of this is related to *token* passing. The CO can send the *token* to MU_H, which in turn can send it to the new CO during registration. The steps, which differ from the first version of TCOT, are:

- MU_H forwards T_i to the CO.
- The CO fragments T_i, computes $E_t's$ of all the fragments, creates *tokens*, and sends them to the members of the commit set. (This step uses one extra downlink message)
- MU_H computes S_t for its fragment.

7.16.6 Correctness

A commit decision by a CO is said to be "correct" if the decision to commit is unanimous. Suppose the CO decides to commit T_i when at least one member of the commit set is undecided. This is possible only if the CO declares commit before the

expiration of either S_t or absence of commit message from at least one DBS. This, however, cannot happen. Further, suppose that the MU_H failed and could not send updates to the CO within S_t or the "commit message" is not received by the CO. In this situation, the CO will abort T_i. Since our algorithm is based on timeout, it is not possible that at any stage the CO will enter into an infinite wait.

7.17 SUMMARY

This chapter introduced a reference architecture of mobile database system and discussed a number of transaction management issues. It identified a number of unique properties of mobile database system and discussed the effect of mobility on its functionality.

It demonstrated that location of the database and the location of the origin of the query must be considered to enforce ACID properties of transactions. To handle these requirements, the concept of *Location Dependent Data* and *Location Dependent Commit* were introduced. Thus in mobile database systems a user initiates (a) a location-dependent query, (b) location-aware query, or (c) a location-independent query. The concept of *data region* was introduced to accommodate cellular structure in mobile database processing and transaction commit.

It identified unique system requirements for concurrency control mechanisms and transaction commitment. First it analyzed the relationship between mobility and transaction processing. A clear understanding of this relationship is necessary for the development of mobile transaction model and its management.

It argued that conventional 2-phase or 3-phase commit protocols were not suitable for mobile database systems and illustrated that a commit protocol which uses least number of messages and offer independent commit decision capability was highly desirable. It introduced one-phase commit protocol with above properties.

A data replication scheme for connected and disconnected operations was discussed for mobile database system. Under this scheme, data located at strongly connected sites are grouped in clusters. Bounded inconsistency was defined by requiring mutual consistency among copies located at the same cluster and controlled deviation among copies at different clusters. The database interface is extended with weak operations.

This chapter provided necessary material for the development of mobile database system framework and mobile transaction model.

Exercises

1. Highlight the essential differences of mobile database system with conventional database systems. What are the problems in using mobile units or a base station or a fixed host as (a) a client, (b) a server, or (c) a peer?

2. Consider the architecture of a given mobile database system. What types of scenario a transaction may encounter during its execution? Explain your own ideas in managing these situations successfully.

3. Develop your own mobile transaction model and a way of executing them on a mobile database system.

4. Implement a strict two-phase locking mechanism in a mobile database system and count the total number of messages it requires to (a) commit a transaction, (b) roll-back a transaction, and (c) execution a transaction (not commit).

5. Consider modifying TCOT to manage transaction failure more efficiently.

REFERENCES

1. R. J. Abbott and Hector Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 17, No. 3, 1992.

2. Alonso, R., D. Barbara, and H. Garcia-Molina. "Data Caching Issues in an Information Retrieval System," *ACM Transactions on Database Systems*, Vol. 15, No. 3, September 1990.

3. Barbará, Daniel "Certification Reports: Supporting Transactions in Wireless Systems". In *ICDCS*, 1997.

4. P. A. Bernstein, V. Hadzilacos and N. goodman, "Concurrency Control and Recovery in Database Systems," Adison-Wesley, Reading, MA, 1987.

5. P. K. Chrysanthis, "Transaction Processing in a Mobile Computing Environment," in *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, Princeton, NJ, Oct. 1993.

6. S. B. Davidson, H. Garcia-Molina, and D. Skeen. "Consistency in Partitioned Networks," *ACM Computing Surveys*, Vol. 17, No. 3, September 1985.

7. M. H. Dunham, A. Helal, and S. Baqlakrishnan. "A Mobile Transaction Model that Captures Both the Data and the Movement Behavior," *ACM/Balter Journal on special topics in mobile networks and applications*, Vol. 2, No. 2, 1997.

8. G. H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer*, Vol. 27, No. 6, April 1994.

9. H. Garcia-Molina and G. Wiederhold, "Read-Only Transactions in a Distributed Database," *ACM Transactions on Database Systems*, Vol. 7, No. 2, June 1982.

10. H. Garcia-Molina and K. Salem, "Sagas," in *Proceedings of ACM SIGMOD Conference*, 1987.

11. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Francisco, 1993.

12. J. Gray, P. Helland, P. E. O'Neil, and D. Shasha, "The Dangers of Replication and a Solution," in *Proceedings of ACM SIGMOD Conference*, 1996.

13. A. K. Elmagarmid, Y. Lie, and M. Rusinkiewicz, "A Multidatabase Transaction Model for INTERBASE," in *International conference on Very Large Databases (VLDB)*, Brisbane, Australia, Aug. 1990.

14. L. B. Huston and P. Honeyman, "Partially Connected Operation," *Computing Systems*, Vol. 4, No. 8, Fall 1995.

15. T. Imielinksi and B. R. Badrinath, "Wireless Mobile Computing: Challenges in Data Management," *Communications of the ACM*, Vol. 37, No 10, Oct. 1994.

16. J. J.Kistler and M. Satyanarayanan, "Disconnected Operations in the Coda File Systems," *ACM Transactions on Computer Systems*, Vol. 10, No. 1, Feb. 1992.

17. H. Korth, E.Levy, and A. Silberschatz, "A Formal Approach to Recovery by Compensating Transactions," in *Proceedings of the 16th VLDB Conference*, Brisbane, Australia 1990.

18. N. Krinshnakumar and A. J. Bernstein, "Bounded Ingnorance: A Technique for Increasing Concurrency in a Replicated System," *ACM Transactions on Database Systems*, Vol. 19, No. 4, Dec. 1994.

19. V. Kumar and M. Hsu, "A Superior Two-Phase Locking Algorithm and its Performance," *Information Sciences*, Vol. 54, No. 1–2, 1991.

20. V. Kumar, M. H. Dunham and N. Prabhu, "A Mobile Transaction Framework Supporting Spatial Replication and Spatial Consistency," *Special Issue on Mobile Databases International Journal of Computer Systems Science & Engineering*, Vol. 20, No 2, March 2005.

21. V. Kumar, N. Prabhu, M. H. Dunham, and A. Yasemin Seydim, "TCOT - A Timeout-based Mobile Transaction Commitment Protocol," *Special issue of IEEE Transaction on Computers*, Vol. 51, No. 10, Oct. 2002.

22. V. Kumar, "A Timeout-based Mobile Transaction Commitment Protocol," in *2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems*, in cooperation with ACM SIGMOD-Moscow, Sep. 5-8, 2000, Prague, Czech Republic.

23. V. Kumar and M. Dunham, "Defining Location Data Dependency, Transaction Mobility and Commitment," Technical Report 98-cse-1, Southern Methodist University, Feb. 98.

8

Mobile Database Recovery

This chapter deals with recovery in a mobile database system, which is more complex compared to conventional database recovery. It first introduces fundamentals of database recovery and briefly describes conventional recovery protocols and uses them to focus on application recovery where information gathering and their processing for recovery is quite complex. The chapter first identifies those aspects of a mobile database system which affect recovery process. It then discusses recovery approaches which have appeared in the literature. Similar to other areas such as transaction modeling, concurrency control, etc., database recovery is also in the development stage, so the coverage here is mostly limited to state -of-the art research and little on commercial products. A number of recovery schemes have been developed [3, 5, 11, 15, 17, 18, 19, 20, 22, 23, 24], and this chapter discusses a few of them.

8.1 INTRODUCTION

Database recovery protocols recover a database from transaction or system failures, that is, they restore the database to a consistent state from where transaction processing resumes. These failures may occur due to a number of reasons such as addressing error, wrong input, RAM failure, etc. In a concurrent execution environment when a failure occurs then a transaction may be active or blocked or being rolled back or in the middle of a commit. The task of a recovery protocol is to identify the right operation for for recovery for each transaction. These operations are (a) *Roll forward* or *Redo* and (b) *Roll backward* or *Undo*. Depending upon the execution status of a transaction, one of these operations is selected. Thus, in a recovery process some transactions are

undone and some transactions are redone. To implement these operations, *Transaction log* is required, which is generated and maintained by the system. The log contains committed values of data items (Before Image – BFIM) and modified values of data items (After Image – AFIM). The log is a crucial document for recovery; therefore, it is generated and maintained by a protocol called *Write Ahead Logging – WAL.* The protocol guarantees that the contents of a log is reliable and can be used for Undo and Redo operations.

After a failure the database system reboots and, by using log, applies Redo and Undo operations on transactions which were in the system when it failed. A Redo completes the commit operation for a transaction, and an Undo rolls back a transaction to maintain atomicity. These operations give us four different recovery protocols: (a) *Undo–Redo*, (b) *Undo–No Redo*, (c) *No Undo – Redo*, and (d) *No Undo – No Redo* [8].

Undo–Redo: This protocol applies Redo and Undo to recover the database systems. This means that during transaction execution it can write to the database intermediate values of its data item. If the transaction was active when the system failed, then the transaction is Undone and it is Redone if the transaction was ready to commit.

Undo – No Redo: This protocol does not support Redo and recovers the database by applying Undo operation only. This means that the system forces intermediate updates of transactions to he database immediately.

No Undo – Redo: This protocol makes sure that no intermediate results of a transaction are installed in the database. Thus, if a transaction cannot be Redone at the time of recovery, then it is removed from the system.

No Undo – No Redo: This protocol does not apply Redo and Undo and recovers the database by using the *shadow* copy of data items. Thus, during execution a transaction creates a show copy of data items it modifies. During recovery it uses actual and shadow copies of a data item to select the right version to install in the database.

Recovery is a time-consuming and resource-intensive operation, and these protocols require plenty of them. The most expensive operation is managing the log. This operation is essential for recovery, so for a Mobile Database System an economical and efficient scheme of its management is necessary.

A *Mobile Database System (MDS)* is a distributed system based on client server paradigm, but if functions differently than conventional centralized or distributed systems. It achieves such diverse functionalities by imposing comparatively more constraints and demands on MDS infrastructure. To manage system-level functions, MDS may require different transaction management schemes (concurrency control, database and application recovery, query processing, etc.), different logging schemes, different caching schemes, and so on.

In any database management system, distributed or centralized, the database is recovered in a similar manner and the recovery module is as an integral part of the database system. Database recovery protocols, therefore, are not tampered with user

level applications. A system which executes applications, in addition to database recovery protocol, requires efficient schemes for *Application recovery* [12, 13]. The application recovery, unlike database recovery, enhances application availability by recovering the execution state of applications. For example, in MDS or in any distributed system a number of activities related to transactions' execution, such as transaction arrival at a client or at a server, transaction fragmentation and and the distribution of these fragments to relevant nodes for execution, dispatch of updates made at clients to the server, migration of a mobile unit to another cell (handoff), etc., have to be logged for recovering the last execution state. With the help of the log the application recovery module recreates the last execution state of application from where normal execution resumes.

Application recovery is relatively more complex than database recovery because of (a) the a large numbers of applications required to manage database processing, (b) presence of multiple application states, and (c) the absence of the notion of the "last consistent state." This gets more complex in MDS because of (a) unique processing demands of mobile units, (b) the existence of random handoffs, (c) the presence of operations in connected, disconnected, and intermittent connected modes, (d) location-dependent logging, and (e) the presence of different types of failure. These failures can be categorized as *Hard failure* and *Soft failure* [17]. Hard failures includes loss of mobile unit (stolen, burnt, drowned, dropped, etc.), which cannot be easily repaired. Soft failures include system failure (program failure, addressing errors, battery ran out, processing unit switched off, etc.) and are recoverable.

An application can be in any execution state (blocked, executing, receiving data slowly, and so on). In addition to this, the application may be under execution on stationary units (base station or database server) or on mobile units or on both. These processing units, especially the mobile unit, may be (a) going through a handoff, (b) disconnected, (c) in a doze mode, (d) turned off completely. The application may be processing a *mobilaction* or reading some data or committing a fragment, and so on. If a failure occurs during any of these tasks, the recovery system must bring the application execution back to the point of resumption.

In application recovery, unlike data consistency, the question of application consistency does not arise because the application cannot execute correctly in the presence of any error. Thus, the most important task for facilitating application recovery is the management of log. The database recovery protocols provide a highly efficient and reliable logging scheme; unfortunately, even with modifications, the conventional logging scheme would impose unmanageable burden on resource constrained MDS. What is needed is an efficient logging scheme, which stores, retrieves, and unify fragments of application log for recovery within the constraints of MDS.

8.2 LOG MANAGEMENT IN MOBILE DATABASE SYSTEMS

Log is a sequential file where information necessary for recovery is recorded. Each log *record* represents a unit of information. The position of a record in the log identifies the relative order of the occurrence of the event the record represents. In

legacy systems (centralized or distributed) the log resides at fixed locations which survive system crashes. It is retrieved and processed to facilitate system recovery from any kind of failure. This persistence property of log is achieved through the protocol called *Write Ahead Logging (WAL)* [4].

This static property of log ensures that no additional operation other than only its access is required to process it for recovery. The situation completely changes in the systems which support terminal and personal mobility by allowing the processing units to move around. As a result they get connected and disconnected many times during the entire execution life of transactions they process. The logging becomes complex because the system must follow the WAL protocol while logging records at various servers.

An efficient applicaion recovery scheme for MDS requires that the log management must consume minimum system resources and must recreate the execution environment as soon as possible after MU reboots. The mobile units and the servers must build a log of the events that change the execution states of *mobilaction*. Messages that change the log contents are called write events [22]. The exact write events depend on the application type. In general, the mobile unit records events like (a) the arrival of a *mobilaction*, (b) the fragmentation of *mobilaction*, (c) the assignment of a coordinator for *mobilaction*, (d) the mobility history of the mobile unit (handoffs, current status of the log, its storage location, etc.), and (e) dispatch of updates from *mobilaction* to DBSs. The DBSs may record similar events in addition to events relating to the commit of *mobilaction*.

8.2.1 Where to Save the Log?

Schemes that provide recovery in the PCS (Personal Communication System) system saves the log at the BS where the mobile unit currently resides [19, 22]. It is important to note that managing log for PCS failure is relatively easy because it does not support transaction processing. However, the concept can be used to develop efficient logging schemes for MDS.

There are three places the log can be saved: (a) MSC (Mobile Switching Center), (b) Base Station (BS), and (c) Mobile Unit (MU). The reliability and availability of mobile units, however, make it a less desirable place to save the log. MSC and BS are suitable places; but from cost and management viewpoints, MSC is not a convenient location. An MSC may control a large number of BSs; in the event of a failure, accessing and processing the log for specific transaction may be time-consuming. An MSC is not directly connected to database servers (Figure 7.1), which provide necessary log management applications. BSs, on the other hand, are directly connected to DBSs and also to mobile units. Therefore, from connectivity and availability aspects, BSs are comparatively better candidates for saving an application log. Under this setup a mobile unit can save log at the current BS and the BS then can archive it on DBSs.

Effect of Mobility on Logging: In conventional database systems, the log generation and its manipulation are predefined and fixed. In a mobile environment, this may

not always be true because of the frequent movements and disconnections of mobile units. A *mobilaction* may be executed at a combination of mobile units, base stations and fixed hosts. Furthermore, if a fragment of *mobilaction* happens to visit more than one mobile unit, then its log may be scattered at more than one base stations. This implies that the recovery process may need a mechanism for *log unification* (logical linking of all log portions). The possible logging schemes can be categorized as follows:

Centralized logging–Saving of log at a designated site: Under this scheme a base station is designated as logging site where all mobile units from all data regions save their log. Since the logging location is fixed and known in advance, and the entire log is stored at one place, its management (access, deletion, etc.) becomes easier. Under this scheme, each mobile unit generates the log locally and, at suitable intervals or when a predefined condition exists, copy its local log to the logging base station. If a fragment or *mobilaction* fails, then the local recovery manager acquires the log from the base station and recover the *mobilaction*. This scheme works, but it has the following limitations:

- It has very low reliability. If the logging base station fails, then it will stop the entire logging process; consequently, transaction processing will stop until the BS recovers. Adding another backup base station will not only increase resource cost but will increase log management cost as well.
- Logging may become a bottleneck. The logging traffic at logging base station may become unmanageably heavy, causing significant logging delays.

For a lightly loaded system with little MU movement, however, this scheme provides a simple and efficient way of managing the log.

Home logging: Every mobile unit stores its log at the base station it initially registers. Although a mobile unit will roam around in the geographical domain freely and continue to access data from any sites, all logging will still be at its base station. This scheme has the following limitations:

- Under this scheme the entire log of *mobilaction* may be scattered over a number of base stations if its fragments are processed by different mobile units with different base stations. To recover the *mobilaction*, all pieces of log will require linking (logically).
- It may not work for spatial replicas (location-dependent data). Consider a location-dependent query which comes to a mobile unit for processing but whose base station is not the one that stores the location dependent data. This may happen if a traveler from Kansas City issues a query on his/her mobile unit for Dallas Holiday Inn data. This scheme can cause excessive message traffic.
- Since the logging location is not distributed, it has poor availability and excessive message traffic during transaction execution.

At a designated base station

Under this scheme a mobile unit locally composes the log and, at some predefined intervals, saves it at the designated base station. At the time of saving the log a mobile unit may be in the cell of the designated base station or at a remote base station. In the latter case, the log must travel through a chain of base stations, ending up at the designated base station. This will work as long as there is no communication failure anywhere in the chain of base stations.

At all visited base stations

In this scheme a mobile unit saves the log at the base station of the cell it is currently visiting. The entire application log is stored in multiple base stations, and at the time of recovery all log portions are unified to create the complete log. It is possible that two or more portions of the entire log may be stored at one base station if the mobile unit revisits the station. A number of logging schemes were developed under these two approaches, some of which are discussed below.

Lazy scheme: In lazy scheme [22], logs are stored on the current base station and if the mobile unit moves to a new base station, a pointer to the old base station is stored in the new base station. These pointers are used to unify the log distributed over several base stations. This scheme has the advantage that it incurs relatively less network overhead during handoff as no log information needs to be transferred. Unfortunately, this scheme has a large recovery time because it requires unification of log portions.

The log unification can be performed in two ways: (a) distance-based scheme and (b) frequency-based scheme. In a distance-based scheme [19] the log unification is initiated as soon as the mobile unit covers the predefined distance. This distance can be measured in terms of base station visited or in terms of cell site visited. In the frequency-based scheme [19], log unification is performed when the number of handoffs suffered by the MU increases above a predefined value. After unifying the log, the distance or handoff counter is reset.

Pessimistic scheme: In the pessimistic scheme [22], the entire log is transferred at each handoff from old to new base station. This scheme, therefore, combines logging and log unification. Consequently, the recovery is fast, but each handoff requires large volumes of data transfer.

The existing mobile network framework is not efficient for full-fledged database transactions running at DBSs and mobile units. In the above schemes the location change of MU has to be updated by DBSs, which would be a big disadvantage. To overcome this, mobile IP was introduced. In Ref. [25] log recovery based on the mobile IP architecture is described where base stations store the actual log and checkpoint information and the base station or the *home agent* as defined in Ref. [21] maintains the recovery information as the mobile unit traverses. This scheme has the advantage that log management is easy and the database servers need not be

concerned with the mobile unit's location update, but it suffers when the mobile unit is far away from home. Consequently, recovery is likely to be slow if the home agent is far from the mobile unit. The other problem with using mobile IP is triangular routing where all messages from the database server to the mobile unit have to be routed through the home agent. This invariably impedes application execution. The schemes discussed so far do not consider the case where a mobile unit recovers in a base station different from the one in which it crashed. In such a scenario, the new base station does not have the previous base station information in its VLR (Visitor Location Register), and it has to access the HLR (Home Location Register) to get this information [8], which is necessary to get the recovery log. HLR access may increase the recovery time significantly if it is stored far from the MU. A similar disadvantage can be observed in the mobile IP scheme of Ref. [25], where the mobile unit needs to contact the home agent each time it needs recovery.

8.3 MOBILE DATABASE RECOVERY SCHEMES

In this section a number of recovery schemes have been discussed. These schemes take different approaches; however, they build their scheme on same mobile database platform. The platform contains a set of mobile unites and base stations. These units save logs and checkpoint necessary activities and make sure that necessary information is available for recovering from failure efficiently and economically.

8.3.1 A Three-Phase Hybrid Recovery Scheme

A three-phase checkpointing and recovery scheme is discussed in Ref. [11] which combines coordinated and communication-induced checkpointing schemes. All base stations use coordinated checkpointing, and the communication-based checkpointing is used between mobile units and base stations. Following steps briefly describe the working of the algorithm. Further details can be found in Ref. [11]. The algorithm uses mobile units MU_1, MU_2, MU_3, and MU_4, as well as base stations MSS_1, MSS_2, and MSS_3, for describing message traffic.

- Initially, a coordinator (base station) MSS_1 broadcasts a request message with a checkpoint index to MSS_2 and MSS_3.
- Each MSS sets up a timer T_{lazy}. It uses a lazy coordination scheme to reduce the number of messages, therefore, it is especially suitable for mobile database systems. In this approach, infrequent snapshots are taken which only occasionally impose high checkpoint overheads of coordinated snapshots on the low-bandwidth network connecting all mobile units. This approach also prevents the global snapshot from getting out of date; as a result, the amount of computation for recovery from failure is minimized.
- Mobile unit MU_2 or MU_3, whichever is active, takes a checkpoint before message m_2 or m_3 arrives from MSS_2 or MSS_3 during T_{lazy}.

- MU_1 or MU_4 takes a checkpoint when T_{lazy} has expired, and it receives a checkpoint request from MSS_1 or MSS_3.
- MSS_2 and MSS_3 responds (send a response message) to MSS_1.
- MSS_1 broadcasts a commit message to all $MSSs$ after receiving response messages from other base stations.
- MU_3 migrates from MSS_3 to MSS_2 and sends a message to wake MU_4 if it is in doze mode.
- MU_2 takes a checkpoint before it disconnects itself from the network. If MU_2 is already in disconnected mode, then it does not take any checkpoint.
- In case MU_1 fails, it stops executing and sends a recovery message to MSS_1.
- MSS_1 broadcasts a recovery messages to all $MSSs$.
- Each MSS sends recovery message to all its MUs. These MUs roll back to their last consistent state.

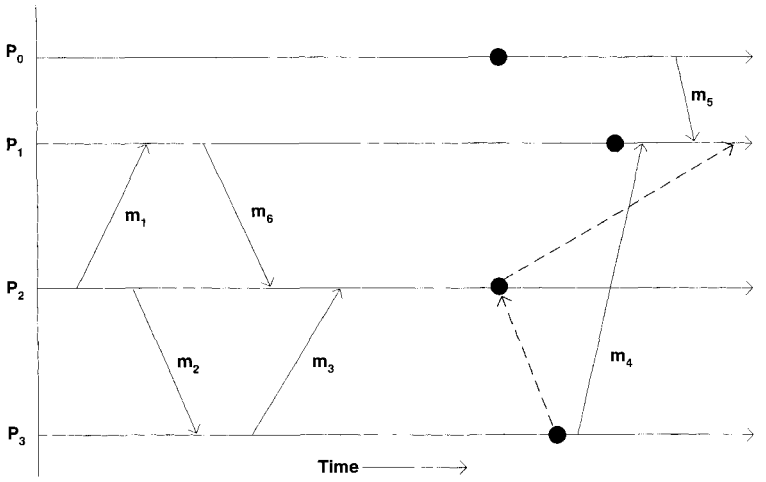

Fig. 8.1 An example of snapshot generation.

8.3.2 Low-Cost Checkpointing and Failure Recovery

In Ref. [23] a low-cost synchronous snapshot collection scheme is presented in which allows minimum interference to the underlying computation. The working of the algorithm is explained with the following example. Figure 8.1 illustrates the flow of messages which manage the snapshot process. The processing noes are represented as P_0, P_1, P_2, and P_3, and m_1, m_2, m_3, m_4, m_5, and m_6 represent the messages.

- The node P_2 first collects local snapshots at the point X time point.
- Assume that nodes P_1, P_3, and P_2 are dependent, so a snapshot request message is sent to P_1 and P_3 by P_2. Node P_3 sends message m_4 to node P_1 after taking its own snapshot.
- Their are two possibilities when message m_4 reaches P_1: (a) P_1 has not processed any message since its last local snapshot or (b) P_1 has already processed a message from any node since its last snapshot. In this example, since P_1 has not processed any message, as a result it takes its tentative snapshot and records this event before processing message m_4. It then propagates the snapshot.
- Node P_0 takes a local snapshot since it has not received any message from any node and sends a message m_5 to P_1. When m_5 reaches P_1, it finds that m_5 is not a new message to force a snapshot so P_1 does not take a snapshot.

When a node P_i fails, then it rolls back to its latest checkpoint and sends rollback requests to a subset of nodes. When a node P_j receives its first rollback message, then (a) it rolls back to its latest checkpoint and (b) it sends a rollback request to a selective set of nodes. Node P_j may receive subsequent rollback messages as a result of P_1's failure, but it ignores all of them. In the case of mobile units, all their rollback requests are routed through their base stations.

8.3.3 A Mobile Agent-Based Log Management Scheme

Mobile agents have been successfully used in managing a number of application and system activities. It has also been used to develop a scheme to manage an application log in MDS (Mobile Database Systems). A mobile agent is an autonomous program that can move from machine to machine in a heterogeneous network under its own control. It can suspend its execution at any point, transport itself to a new machine, and resume execution from the point it stopped execution. An agent carries both the code and the application state. Actually a mobile agent paradigm is an extension of the client/server architecture with code mobility. Some of the advantages of mobile agents as described in Ref. [14] are:

- **Protocol Encapsulation**: Mobile agents can incorporate their own protocols in their code instead of depending on the legacy code provided by the hosts
- **Robustness and fault-tolerance**: When failures are detected, host systems can easily dispatch agents to other hosts. This ability makes the agents fault-tolerant.
- **Asynchronous and autonomous execution**: Once the agents are dispatched from a host, they can make decisions independently and autonomously. This is particularly useful to the wireless environment where maintaining a connection throughout an executing *mobilaction* may not be economical or necessary. In such cases, the agents can visit the destination, perform any required processing,

and bring the final data to the origin thereby removing the need for a continuous wireless connection. For example, an agent can take a *mobilaction* from a mobile unit, execute it at the most suitable node (could be remote), and bring the result back to the mobile unit.

Agents do have disadvantages, and the one which is likely to affect the logging scheme is its high migration and machine load overhead [2]. This overhead must be minimized for improving the performance. The present scheme uses agent services with the *only when needed approach.* It is not possible to develop a scheme, which optimizes the performance at all levels and in all different situations. For this reason, some recovery schemes improve the performance by targeting to minimize the communication overhead, some might concentrate on total recovery time, some may optimize storage space, and so on. Thus, each scheme involves certain tradeoffs. When these issues are taken into consideration, it becomes necessary to build a framework that supports the implementation of the existing schemes and should also be able to support any new scheme. The framework should support the activation/deactivation of a scheme, depending on the particular environment in which it offers best performance. Such a framework should abstract the core base station software (which handles the registration, handoff, etc., activities) from handling the recovery procedures, thus allowing for better recovery protocols to be implemented without the need for changing the core software. The framework may also support a rapid deployment of the recovery code without much human intervention.

In MDS, the coordinator module resides in the base station. It splits *mobilaction* into fragments if necessary, and it sends some of them to a set of DBSs. This requirement asks for specific intelligence to be embedded in the base station code. *Mobilaction* initiated by mobile unit may use different kinds of commit protocols like 2-phase commit or 3-phase commit or TCOT (Transaction Commit on Timeout) [9]. The coordinator module needs to support all of these. If such a module at a base station does not support a particular protocol, then there should be an easy way to access such a code. An extension to this is that, when a new efficient protocol is introduced, all base stations should be able to upgrade to this as easily as possible and with little or no human intervention. From the perspective of mobile unit log recovery, an architecture is required which supports intelligent logging and is able to incorporate any future developments without any difficulty.

Some recovery schemes specify that the logs move along with the mobile unit through a multitude of base stations. The new base stations should be able to handle the logs in the same way as the previous one did or log inconsistency might result. It is argued that the flexibility and constraints mentioned above could be successfully incorporated on a mobile-agent based architecture under which the code necessary for recovery and coordination can be embedded in the mobile agents. The coordinator can be modeled as a mobile agent and can be initiated by the mobile unit itself if necessary. If during a handoff the new base station does not support a specific logging scheme, then the agent in the previous base station which supports this can clone itself and the new replica can migrate to the current base station without any manual intervention. The same technique can be used in quickly populating the base stations with any new

protocols. The mobile agent with the new protocol embedded in it can be introduced in any base station and it can replicate and migrate to other base station.

8.3.4 Architecture of Agent-Based Logging Scheme

An architecture is presented where mobile agents are used to provide a platform for managing logging. The architecture supports the independent logging mechanisms. It is assumed that each base station supports the functionality of mobile agents. The main components of the architecture are:

Bootstrap agent (BsAg): This agent handles a base station failure. Any agent that wishes to recover should register with the bootstrap agent. The base station initiates the bootstrap agent. Once loaded, this agent starts all the agents that have registered with it. These agents have the capability to read the log information they have created and act accordingly. The need for such an agent may be obviated if the mobile agent provides an automatic revival of the agents with their state intact.

Base Agent (BaAg): This agent decides which logging scheme to use in the current environment. Such functionality can be decided by its own intelligence or can be given as an input. For every mobile unit, it creates an instance of an agent that handles the recovery of *mobilactions* based on the relevant logging scheme.

Home Agent (HoAg): This agent handles *mobilactions* for each mobile unit. It is responsible for maintaining log and recovery information on behalf of the mobile unit. The mobile unit sends log events to this agent, which is responsible for storing them on the stable storage of the base station. The HoAg is a base station interface to the mobile unit for Mobilactions

Coordinator Agent (CoAg): This agent resides at base station and acts as the coordinator for all *mobilactions*.

Event Agent (EvAg): In addition to the above framework, the base station provides mobile agents with an interface to the various events taking place like registration of a mobile unit, failure of a mobile unit, handoff of a mobile unit, etc. This approach abstracts away the core base station functions from application recovery support. When a mobile unit suffers handoff, its HoAg should know about it so that it can perform the required operations. The EvAg is the interface for the base station to the agent framework for dissemination of such information.

Driver Agent (DrAg): The migration of a mobile agent during a handoff involves the movement of its code and the actual data. This might generate considerable overhead [2] even if the actual log transfer is not much.

8.3.5 Interaction Among Agents for Log Management

These agents collaborate with each other to facilitate log management.

Interaction of CoAg and HoAg: An MU sends Mobilaction to its HoAg, which forwards it to the corresponding CoAg. If the CoAg needs to contact the MU, it does so through the MU's corresponding HoAg. When CoAg sends a write event to the HoAg, it stores it in its local store before sending it to the MU. Similarly if any events come to the MU through user input, MU sends the corresponding log messages to the HoAg.

Action of agents when handoff occurs: The HoAg moves along with the mobile unit to the new base station in a handoff. Based on schemes like Lazy and Frequency-based, the agent may or may not take the stored logs along with it to the new base station. When a handoff occurs, a driver agent (DrAg) is sent along with the necessary log information to the new base station instead of the whole HoAg with all its intelligence for log unification. The DrAg has a very light code whose main function is to see whether the code for HoAg is present in the new base station. If so, it requests the resident BaAg in the new base station to create an instance of the HoAg for the mobile unit. If any compatible code is not present, then the DrAg sends a request to the previous base station's BaAg, which clones the necessary HoAg and sends the copy to the new base station. When the mobile unit moves out of a base station, its log information is not deleted automatically but it is stored unless notified otherwise by the agent of the mobile unit. This facilitates the unification of logs when logs are distributed over a set of base stations.

8.3.6 Forward Strategy

All schemes reviewed earlier have assumed instant recovery of the mobile unit after a failure, but Ref. [8] acknowledges the possibility where the mobile unit might crash in one base station and recover in another base station. A time interval is defined between the mobile unit failing and its subsequent rebooting as Expected Failure Time (EFT). This scheme concentrates on such scenarios where the EFT is not so trivial that the recovery occurs instantaneously. Base station detects the failure of a mobile unit and agents do not play any part in such detection. For example, if the communication between two mobile units breaks down because of the failure of one of the mobile units, then the corresponding BS will immediately know about this event. Similarly, base station also knows which mobile unit has executed power-down registration, which mobile unit has undergone a handoff, and so on.

A base station also continuously pages its mobile units.[1] If the mobile unit suffers a handoff, then the communication with the last base station is not broken until

[1]Sprint PCS system pages its mobile units after every 10 to 15 minutes without generating any overhead to learn their status, and a mobile unit also continuously scans the air by using its antenna to detect the strongest signal.

the connection with the new base station is established (soft handoff). These features of PCS allow MDS to detect mobile unit failure. Thus, while a mobile unit is executing its fragment, its status is continuously monitored by the base station and any change in mobile unit's situation is immediately captured by the Event Agent interface. Since this detection is system-dependent, EFT (Expected Failure Time) tends to be an approximate value. The detection can be passed on to the HoAg in many ways. The MDS can provide an interface, which would allow the agents to wait for an event. Another approach would be to provide an agent readable system variable which would be set on any such event. The agent will periodically poll the variable to check if it is set. Both approaches are possible and easy to implement in languages such as Java in which many agent systems like IBM's Aglets and General Magic's Odyssey have been developed [6]. Since handoff does not occur in the above case as pointed out in ref. [8], the new base station does not know the location of the old base station. This situation leads to the new base station contacting the Home Location Register (HLR) for the previous base station [8, 18, 19, 23]. This might be a hindrance to fast recovery if the HLR happens to be far from the querying base station. Actually the Visitor Location Register (VLR) is first queried for the previous base station information, which is stored in VLR if both base stations happen to fall under the control of the same VLR. If base stations are under different VLRs, then the HLR of the mobile unit has to be queried. Such information is stored in the HLR when a mobile unit first registers with a base station.

In the lazy scheme [8], the base station starts building up the log immediately upon failure of mobile unit. In the schemes presented in Ref. [19], the mobile unit explicitly issues a recovery call to the base station and the base station begins the log unification. This raises certain questions in the event of the mobile unit crashing and recovering in a different base station. If the log is to be unified immediately upon a failure, then it might be necessary for the new base station to wait for the old base station to finish its unification and then present its log. If the failure time is large or the total log size is small, then unification will be over by the time the new base station queries the previous base station. In such a case, recovery can be fast. In the case of a relatively small *EFT (Expected Failure Time)* or a large log size (to be unified), the new base station must wait first for the unification and then for the actual log transfer. This results in increased recovery time and network cost. In such cases it might be preferable for the log unification to be done in the new base station if the list of base stations where the log is distributed is known. Such a list is transferred in schemes provided in Ref. [19] and not for those in Ref. [8]. In the approach where the log is unified after a recovery call, the recovery time might not be small enough if the log size to be unified is small. In this case the unification has to begin after getting the list of base stations involved from the previous base station. Also, if the mobile unit has not migrated to a new base station before recovery, then the log has to be unified, which is likely to increase the recovery time.

Reducing Recovery Time

The scheme of log unification is based on the number of handoffs occurred since the last log unification or the start of the transaction whichever is later. The log is unified periodically when the number of handoffs occurred crosses a predefined handoff-threshold.

When a handoff occurs, the *Trace* information is transferred from the old base station to the new base station. This trace information is an ordered list of elements giving information about the base stations involved in storing mobile unit's log. Each array element consists of two values: (a) the identify of this base station (BS-ID) and (b) the size of the log stored at BS-IDi (Log-Sizei). When a handoff occurs, then BS-ID of the new base station and a Log-Size value of zero are added to the end of the trace. The Log-Size value is updated whenever mobile unit presents base station with some log information. Optional parameters can also be present in the trace information. Since the trace does not contain the actual log contents and is mostly an array of base stations identities and log sizes, it does not present a significant overhead during the handoff. The scheme also assumes the presence of *EFT (expected failure time)* value which can be stored as an environment attribute accessible to HoAg of the mobile unit at the base station. If such support cannot be given by the system, then HoAg can also estimate *EFT* from mobile unit's activities. If the agent estimates the *EFT*, then this value is also stored in the trace information. When the system detects mobile unit failure, it informs the agent framework through the *Event Agent* interface. This agent notifies the appropriate HoAg that starts the *EFT* clock. This clock is stopped to get the *Recorded-EFT* value, when the HoAg receives mobile unit recovery call, which can come from the mobile unit in the same base station or from a different base station in which the mobile unit has recovered. In either case, the agent residing in base station where the *EFT* clock is started. It estimates the new *EFT* as

$$\textbf{(K1} \times \textbf{Recorded-EFT)} + \textbf{(K2} \times \textbf{EFT)}, \text{ where } K1 + K2 = 1$$

The new *EFT* is a weighted sum of the previous *EFT* and the *Recorded-EFT*. K1 indicates the reliance on the *Recorded-EFT*, while K2 indicates the reliance on the previously calculated *EFT*. The values of K1 and K2 are functions of the environment. In a network where the failure time is relatively stable, K2 is given more weight; and in a network where the failure time varies frequently, K1 can be given more weight. To improve storage utilization, unnecessary records from the log is deleted. This garbage collection is optional and is done upon log unification. When a mobile unit log is unified at a base station, a garbage-collect message is sent to all the base stations hosting the mobile unit logs as specified in the trace *BS-ID* list. The previous base stations purge these logs on receiving this message. The *BS-ID* and the Log-Size lists are erased from the trace information at the current base station to reflect the unification, and a single entry is created in the trace with the current base station identity and the unified log size.

8.3.7 Forward Log Unification Scheme

Since the trace information contains the size of the log stored at different base stations, the HoAg can estimate the time for log unification based on the network link speed and the total log size. This time is called the *Estimated Log Unification Time (ELUT)*, which can be measured as: *Max (BSi-Log-Size/Network link Speed + Propagation Delay)*, for all base stations in the trace. The exact characterization of the *ELUT* value depends other factors such as whether base stations are located in the same VLR area or different areas, queuing delay, etc. The HoAg should take into consideration as many parameters available from the system as possible to estimate the *ELUT* accurately. Log unification is started if $(\delta * ELUT) \leq EFT$ or else it is deferred until a recovery call is heard from the mobile unit.

The Unification factor "δ" describes what fraction of the log unification will be done by the time the failure time of the mobile unit comes to an end. The default value can be kept as 1, which indicates that the log unification starts only if it can be totally completed by the time the mobile unit is expected to complete its reboot. If the mobile unit reboots in a different base station while the log is being unified in the previous base station, it has to wait for the unification to complete. Variations of this scheme are possible if the HoAg can estimate the effective handoff time. Based on this value, if there is still a long time for the next handoff, then the log unification can start immediately upon a failure, as it is more probable that the failed mobile unit will recover in the base station where it failed rather than in any other base station. In the event the log unification is not performed because $(\delta \times ELUT) \leq EFT$, the HoAg waits for the mobile unit to recover. If the recovery happens in the same base station, then the log unification starts; but if the mobile unit reboots in a different base station, then the HoAg transfers the trace information and the log stored at this base station when requested. In this case, the new base station has to perform the log unification after getting the trace information from the previous base station. This trace contains the newly calculated *EFT* value.

8.3.8 Forward Notification Scheme

This scheme addresses the issue of time spent in getting the previous base station information from the HLR. To minimize this time, a scheme involving forward notifications is proposed. When a mobile unit fails in a particular base station and if the actual failure time (total duration before mobile unit is rebooted) is not too high, then there is a high probability that the mobile unit will recover in the same VLR or in a BS that is in adjacent VLRs. Thus a VLR and its adjacent VLRs cover a large area, and the situation where the mobile unit reboots in a nonadjacent VLR does not occur frequently. If the mobile unit happens to restart in a non-adjacent VLR, then it must have been extremely mobile and most of the recovery schemes are not designed for such unrealistic situation. The other implication is that the mobile unit had been in the failed state for a longer period and so it is likely that the coordinator could have decided to abort the *mobilaction*. Each VLR also stores mobile unit's status information (normal, failed, and forwarded).

When a mobile unit fails, its corresponding HoAg informs the VLR about this failure. The VLR first changes the status of the mobile unit in its database from normal to failed. The VLR then issues a message containing its own identity (e.g., identity of the VLR that sends this message), the identity of the failed mobile unit, and the identity of the we propose in which the mobile unit crashed to its adjacent VLRs that the mobile unit has failed. The adjacent VLRs store these messages until explicit denotify messages are received. The mobile unit is recorded in these adjacent VLRs with the status as forwarded. The following scenarios may arise when the mobile unit reboots:

Case 1–The mobile unit reboots in the same base station where it crashed: In this scenario, the HoAg informs the VLR that the mobile unit has recovered. The VLR then issues a denotify message to all the adjacent VLRs indicating that the forward notification information is no longer valid. The status of the mobile unit is changed back to normal from failed.

Case 2–The mobile unit reboots in a different base station but in the same VLR: First the mobile unit registers with the base station and the registration message is logged on to the corresponding VLR. This VLR identifies the status of the mobile unit as failed, and then it proceeds as in case 1 and sends denotify messages to the adjacent VLRs. The status of the mobile unit is changed back to normal from failed. The new base station then proceeds to perform log unification from the previous base station.

Case 3–The mobile unit reboots in a different base station and a different VLR: The mobile unit requests for registration. The corresponding VLR identifies the mobile unit as a forward notified mobile unit and returns the identity of the previous base station and the identity of the VLR to the HoAg of the mobile unit in the recovered base station. The base station then proceeds to perform log unification from the previous base station. Simultaneously, the new VLR sends a recovered message to the previous VLR regarding the recovered status of the mobile unit and also sends a registration message to the HLR regarding the registration of the mobile unit in the new location. The status of the mobile unit is changed to normal from forwarded in the new VLR. Upon receiving the recovered message, the previous VLR sends a denotify message to all adjacent VLRs except the one in which the mobile unit recovered and removes the registration of the mobile unit from itself as well. In the situation where the mobile unit recovers in a nonadjacent VLR that has not received the forward notifications, the new base station has to get the previous base station information from the HLR and then send the previous VLR a recovered message. Upon receiving this message, the previous VLR acts similar to the previous VLR of case 3. The forward notification scheme is unsuitable if the mobile unit suffers failures with a very small *EFT*. In that case the mobile unit recovers in the same base station where it failed. Hence, the forward notifications and subsequent denotifications generate communication overhead. To alleviate this, we might delay the sending of these notifications immediately on failure of the mobile unit. The