# Context Sensitive Database Summarisation

by

Darin Chan, *B.Eng.(IT&T)(Hons)*
School of Informatics and Engineering,
Faculty of Science and Engineering

21st October 2005

A thesis presented to the
Flinders University of South Australia
in total fulfillment of the requirements for the degree of
Doctor of Philosophy

Adelaide, South Australia, 2006

# Table of Contents

# List of Figures

# List of Tables

# Abstract*

The use of databases is a common practice for storing and organising large amounts of data. In a distributed environment, especially widely distributed systems, replication provides a means for storing an entire database locally. However, in an environment where mobile devices are used, resource limitations on both storage and power capacity eliminates the possibility of full replications. Moreover, reliable access to centralised databases is not always possible in these environments. Consequently, there is a need to find ways of storing data on a mobile computer in such a way as to maximise user access while providing answers at an acceptable level of accuracy.

This dissertation argues that it is useful to know and incorporate the user context in which the database will be used in the creation process of the mobile database. It investigates a context sensitive summarisation technique and provides a proof of concept prototype, COSMOS, which maximises data availability based on the context in which the system is being used and the accuracy required by the user. More specifically, a framework for summarising data using the context of the user is introduced.

*Local nulls* are proposed to the relational algebra and SQL to allow efficient querying of summary databases. Some modifications to the database management system include the handling of transactions in the system, update propagation and system failure protocols. These modifications are proposed to allow the effective use of the summarised data.

*This research was partially supported by Australian Research Council SPIRT Grant C00107533, held in collaboration with HP Labs.

# Certification

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

As requested under Clause 14 of Appendix E of the *Flinders University Research Higher Degree Student Information Manual* I hereby agree to waive the conditions referred to in Clause 13(b) and (c), and thus

- Flinders University may lend this thesis to other institutions or individuals for the purpose of scholarly research;
- Flinders University may reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signed Dated

Darin Chan

# Acknowledgements

I would like to thank my supervisor, Professor John Roddick, for his invaluable help in getting me through my candidature.

To my colleagues at the Flinders University of South Australia, School of Informatics and Engineering, I would like to say a grateful thank you. In particular, to Denise de Vries and Aaron Ceglar for their help and encouragement throughout my candidature. I would also like to thank the academics and staff of that school for their continual support over the past few years.

Flinders University of South Australia, in collaboration with HP, had funded my research for the past few years. I would also like to extend my thanks to all the people who were involved there.

Finally, I would like to thank all my friends and family who had helped me during the period of my candidature. In particular, to my parents for their love, support and their confidence in me, and to Adeline Ong for her invaluable help and encouragement.

Darin Chan
October 2005
Adelaide.

# Chapter 1

# Introduction

Mobile systems offer the opportunity to access information, such as those available in a database, without the need of fixed wired connections to the server. Instead, access is typically achieved through the use of wireless network interfaces. This can free the user from the physical constraints of accessing and processing data in limited locations. However, to maximise this advantage, new techniques must be developed as the naive approach provides for a non-optimal solution.

The COntext Sensitive MObile Summarisation (COSMOS) database system discussed in this dissertation is proposed to provide summarisation of a large database based on the user's context. This will shift the reliance on a server database to the local database and effectively maximise the use of local data. Some modifications are required of the database management system. In particular, *local nulls* are proposed to facilitate the querying of data, and other modifications are required to the handling of transactions, update propagation and the failure of important sites in the system.

This chapter introduces mobile systems and outlines the basis of this dissertation. Section 1.1 presents a history of mobile computers. Section 1.2 discusses the limitations of mobile systems. The goal of this dissertation is to design mobile databases that are capable of operating even with these resource limitations. The problem statement and a motivating example are presented in Sections 1.3 and 1.4 respectively. The contributions of this dissertation are outlined in Section 1.5. Finally, the structure of this dissertation is given in 1.6.

## 1.1 Mobile Computer Technologies

In 1981, the Osborne Computer corporation released what was considered to be the first true portable computer, the Osborne 1 Personal Business Computer. This model consists of a carrying handle, small 5in display, keyboard, battery pack and floppy disk drives. This was quickly followed in the following year with the release of the Epson's HX-20 notebook computer, consisting of tape drives, full sized keyboard, a 4-line Liquid Crystal Display (LCD) screen, printer and rechargeable batteries. The sales of these computers showed that the portability of these computers were popular with the general population. Compaq released the Compaq Portable computers in early 1983, which were IBM Personal Computer (PC) compatible, ensuring that software written for the IBM PCs would work on the Compaq Portables, further increasing the popularity for portable computers. In the same year, Tandy/Radio Shack released the popular TRS-80 Model 100 designed by Kyocera with integrated full size keyboard and an 8-line LCD screen display. In early 1984, Commodore released the first portable colour computer. It featured a 5 inch colour monitor and floppy drives, but required mains power. The Psion Organiser 1 handheld computer was also released at the same time. It resembled a calculator with a 16 character LCD display, providing calculator functions, calendar, BASIC programming language and utilities packs for science and maths functions.

From 1985 to 1991, major companies such as IBM, Compaq and Apple, developed and released portable computers that improved on previous releases. Portable computers, that at first had suitcase sized design, was progressively changed to have that of the current laptop computer, where the LCD monitor would close over the keyboard. The 1990's also saw the introduction and growth of the Internet allowing portable computers with modem capabilities to access information from around the world.

In 1993, Apple's Newton introduced a pen-based input device, which lead to the first successful Personal Digital Assistant (PDA). Additionally, with further advances to CPU circuitry that reduced the power consumption and increased performance, subnotebooks began appearing which took the appearance of a laptop which could be held in the user's palm, much like a PDA. Popular models included the HP's OmniBook, IBM's ThinkPad and Toshiba's T3400.

Additionally, in 1994, following the pen-based idea, computers using the slate design similar to current Tablet PC were introduced. These computers used the then available processors, such as the x386 microprocessors, running Windows

3.1 with Pen Extensions. Among the companies that introduced the hardware to support the slate design were RCA, EO, NCR, Samsung, Dauphin, Fujitsu, TelePad, Compaq, Toshiba, and IBM.

In 1996, the Palm Pilot saw PDAs became more affordable and thus more accessible to the public. These devices were popular as they were small and lightweight, while providing an intuitive user interface. The functionalities of PDAs then increased to include games, music and Internet capabilities.

In 1999, affordable PDAs with colour display were introduced as was the IEEE 802.11b wireless interface (commonly known as Wireless Fidelity (WiFi)) that enabled wireless networking between mobile computers and existing wired networks.

The Bluetooth technology originally developed in 1997 for mobile telephony, appeared in mobile computers to allow devices with Bluetooth to interact with each other. These devices included other PDAs, laptops, modems, mobile phones and handsfree headsets. From 2000, mobile computers such as laptops and PDAs were generally fitted with wireless technologies, such as Bluetooth and WiFi, to make them truly mobile.

In 2001 to 2002 saw the reintroduction of the slate computers by Microsoft as Tablet PC. It had similar functionality to the current laptops but also included a touch screen and used a pen-based interface. The Tablet PC is essentially a larger version of the PDA that uses similar CPU to that found in current laptops. It is thus capable of running software similar to a laptop. While the uptake of these mobile computers was slow, it was an important advancement for mobile computers.

The history of large scale database management systems can be considered to began with the 1960s Apollo moon landing project. Vast amounts of information to support the project were required to be managed. As a result of that project, a hierarchical database system and a network database system were developed by IBM and General Electric, respectively, in the 1970s. The relational database system was introduced by Codd, later in the 1970s. The commercialisation of relational database systems in the early 1980's saw an increase in the usage of databases within businesses. With the arrival of the Internet in the 1990's, remote access to computer systems with legacy data were introduced through the Client-Server architecture. This would later serve as an architecture from which mobile computers could access databases, (Connolly, Begg & Strachan 1999).

The development of the Third Generation (3G) mobile phone networks, which began in 1999, aimed to have a single standard for voice, data and video communications. While this technology is currently marketed towards mobile telephony sector, it is also suitable for the network of mobile computers because of its high data transfer rate. This technology can allow compliant mobile computers to operate over a much larger area. More details of wireless communication technologies are discussed in Section 2.1.

While there are many advantages of using mobile computers, there are also many limitations to it when compared to fixed location computers. These issues are discussed in the next section.

## 1.2 Mobile Database Issues

There are many issues concerning the effective use of mobile systems, both in respect to current technology and those likely to become available in the near future. For mobile databases the most important of these are:

- the limiting power and bandwidth capacities,
- the limitations on storage capacity, and
- the security and privacy issues created when a computer is in a mobile environment.

Issues concerning security and privacy may cause significant problems in regards to databases in mobile environments and may include the identification and authentication of users and the theft of data during transmission (Heuer & Lubinski 1996, Zaslavsky & Tari 1998). However, these issues are outside the scope of this thesis. This thesis will focus only on the first two of these issues.

### 1.2.1 Power and Bandwidth Capacities

An important limiting characteristic of mobile computers is their finite battery capacity and the low communication bandwidth available and/or affordable (Imielinski & Badrinath 1994, Pitoura & Bhargava 1993). These limits may lead to frequent disconnection due to both the need to reduce connection costs and because of technical limitations (Heuer & Lubinski 1996, Lubinski 2000).

Advancements in battery technology have allowed longer intervals between recharge[1]. However, even with longer life, the high energy costs associated with data transmission will rapidly decrease battery life of a mobile computer (Pitoura & Bhargava 1993, Zaslavsky & Tari 1998). Thus, intentional disconnection may be required through power management software to allow higher priority operations to continue.

In regards to bandwidth, in most situations there are likely to be many mobile users connected to their centralised or distributed database servers and in some locations, wireless coverage may be patchy. The available communication bandwidth in a mobile environment must therefore be shared between each user, thereby restricting the bandwidth available. As a result, networks may experience outages that can cause the mobile device to be subjected to frequent disconnection. Thus, in terms of mobile databases, access between central servers and the mobile system can be unreliable at times (Madria, Mohania & Roddick 1998). For example, the disconnections that may occur can disrupt a query being performed and as a result, prevent or slow query response.

### 1.2.2 Storage Capacity

Limited storage capacity is another important resource limitation of mobile devices (Lubinski 2000, Pitoura & Bhargava 1993). Large multiple hard drive systems, such as those found in common desktop and server computers, cannot be accommodated into a mobile computer, given that they are required to be portable.

Improvements in hard drive technology have made modern hard drives smaller and with a greater capacity. Devices holding 60Gb are currently available for laptops, while comparable to single drive desktop systems, this is still only a fraction of the storage capability of centralised servers that support corporate databases. However, even with the reduced physical size of the modern hard drive, most are still as large as a PDA. Consequently, the storage capacities of laptop-based hard drives are about 500 times more than that utilised by a flash disk emulator[2]. These flash disk emulators may provide attractive energy

[1]In particular, PDAs (such as the iPaQs produced by HP) provide 600-1400 mAH Li-Ion batteries that may last for up to 10-14 hours of continuous usage, while the Lithium-Ion battery packs for laptops may provide continuous usage for 2-10 hours depending on the battery configurations of the laptop.

[2]Currently, a typical capacity for laptop is around 60Gb while flash disks may provide around 128Mb.

consumption and performance, but are still relatively expensive and have limited capacity (Douglis, Kaashoek, Li, Cáeres, Marsh & Tauber 1994). Due to the storage limitation of mobile computers, it is thus difficult (even if it was logically desirable) to create replicas of large databases on such devices, particularly on handheld devices such as PDAs, which are the current focus of our work.

## 1.3 Problem Statement

Available power capacity is a major limitation in the mobile environment that must be considered. That is, energy consumption must not increase significantly such that it reduces the operating time of the mobile device. With reduced storage capacity, mobile devices are limited to smaller databases. Thus, low availability of data may exist, especially when only the local database is accessible. For many cases, mobile computers are typically connected to other fixed computers through wireless networking, which are prone to being unreliable. Thus, any queries over these networks may compromise the query response time.

**The problem addressed in this dissertation is how to provide methods for ensuring the availability of priority data (through summarisation) on a mobile database system, without compromising its operating and query response time.**

If a mobile database is to function during periods of weak or no connections, then data must be available during these times. The goal of this dissertation is to provide a means by which the important information (with regards to the user) is locally available, reducing the need to access the main database through weak connections.

## 1.4 Motivating Example

As motivation, consider the use of mobile database in the medical profession. Within a hospital scenario, medical practitioners and staff often find themselves moving around visiting patients within different wards. With the use of mobile computers, it is possible to provide updated data readily to staff, without the requirement of a physical network connection. The mobile computer then enables staff to read current data regarding patients for diagnosis purposes, and also provide update regarding the condition of patients they are monitoring.

Additionally, outside the hospital scenario, medical practitioners making house calls may also benefit from the use of mobile databases. Before leaving, the medical practitioner's database may be summarised to contain only the required data, such as the patient's details and any recent illnesses within the current month. Once the practitioner leaves the hospital's network coverage, connection to the main database will be expected to be weak or non-existent. The medical practitioner can access locally available data, and any new information regarding the patient are stored on the mobile computer and merged back with the main database when the connection is more reliable.

In this scenario, each user may require different information from the main hospital database. Thus, personalisation is required to define the required summarised mobile database for each individual user. This personalisation of each mobile database, requires the identification of user context. This context can include

- enumerated data that the user has specified,
- information regarding the interest of the user,
- previous requests from the user,
- the schema of the main database,
- inductive information gathered from techniques such as data mining techniques, and
- any time or location information with regards to the user.

More details of this example are given in Appendix B. This scenario will be used as a running example for examples in the later chapters.

This dissertation develops the COntext Sensitive MObile Summarisation (COSMOS) framework, from which a summarised mobile database is created. COSMOS uses the user's context as criteria in order to determine data that are of important. Once created the mobile database may be used in a client/server, and ultimately a hierarchical architecture. One advantage of using the database in such an architecture is that when disconnected the user can rely on the local mobile database, and if connected, the server database can be queried for more accurate results.

## 1.5 Contribution of this Dissertation

The main contribution of this dissertation is the creation of a framework for summarising and allocating data in a hierarchical mobile distributed database system, COSMOS. A hierarchical architecture is a tree structured architecture that uses a central database as its root. The central database acts as the main storage of all accessible data within the system. Child nodes have a client role, while any parent nodes have a server role. Thus, any node having both a parent and children nodes will have both a client and a server role within the system.

COSMOS employs different context sensitive data selection techniques to identify data of importance to the user. The selection techniques, termed *criteria*, are categorised as Primary, Secondary and Tertiary criteria. Primary criteria include enumeration, contextual, previous usage and push-based. Secondary criteria include model-based and induction, while Tertiary criteria include time and spatial-based criteria. COSMOS is capable of using all or some of the criteria in each category. All data selected are weighed against each other and ordered in regards to importance so that data inclusion is from the most important down until the specified database size is reached. The use of a Storage Map (SM) provides a representation of available data within each summarised database.

The use of COSMOS introduces new issues, which include the management of missing values within a summarised database and the inability to follow strict ACID[3] properties of transactions. This dissertation also provides solutions to these issues. For the missing values, a new semantics for the null value, namely *local nulls*, is proposed. Typically, a summary database represents a subset of the main database, and to represent the locally unavailable data, local nulls are used. That is, a local null value in the summary database represents a value that is unavailable locally but may be available if the main database is accessible.

Strict ACID properties for transactions, while suitable for traditional and even many distributed database system, are quite limiting when used in a mobile environment. Thus, the relaxation of these ACID properties, especially the consistency properties, and introduction of transaction operations to suit the relaxed ACID properties are required.

In order to ensure a COSMOS database system operates correctly, different protocols are required. These protocols include:

1. a function to transform the transaction operations to SM,

[3]Atomicity, Consistency preservation, Isolation, and Durability

2. protocols to ensure serialisability,
3. reconciliation protocols,
4. protocols required when an update is submitted at a local summary database,
5. protocols required once it passes the commit phase at the central database, and
6. recovery protocols if the central database is inaccessible.

## 1.6 Dissertation structure

This dissertation is structured as follows: Chapter 2 surveys the area of mobile databases in terms of wireless networking technologies, the architecture used for mobile databases, and the functionalities required for a database to work within a mobile environment. Chapter 3 provides a survey of data summarisation techniques in terms of modifications to the either schema or domain. Data allocation schemes are also examined, in the context of distributed and mobile database systems. Chapter 4, presents a survey of how context is used in mobile computing. Chapter 5 discusses the construction of summary databases using COSMOS. Chapter 6 discusses why local nulls are important and explains how they work in terms of relational algebra and SQL. Chapter 7 discusses transaction management of a COSMOS database system and Chapter 8 provides a list of protocols required by that system. Chapter 9 provides an evaluation of a COSMOS database system. Finally, Chapter 10 provides a conclusion and discussion of future work.

# Chapter 2

# A Survey on Mobile Databases

In contrast to distributed databases operating on traditional hardware, the devices available to mobile databases exhibit severe resource limitations that affect the manner in which data is managed. However, in many respects both conventional distributed databases and mobile databases may be considered as special cases of a common architecture and both can borrow ideas from the other (Dunham & Helal 1995). Indeed, Holliday, Neumann and others (Holliday, Agrawal & Abbadi 2000, Neumann & Maskarinec 1997) discuss the addition of mobile components within a distributed database system. In short, many distributed data management issues are similar to those that affect mobile database systems, and the innovations being considered for mobile databases are equally applicable to some distributed data management problems. This chapter surveys some of these techniques focussing on summarisation techniques for mobile databases. To do this the manner in which database functionality is maintained in mobile database systems is discussed, including critical functions such as query processing, data replication, concurrency control, transaction support and system recovery.

This chapter is structured as follows. Section 2.1 discusses the three major classes of mobile wireless technology. Section 2.2 will then examine the architectures available in a mobile environment in terms of the operational modes experienced by mobile applications, the wireless technologies currently available and the database architectures that may be used. Finally, Section 2.3 explores different techniques for implementing common database functions.

[1]A version of this chapter appears as part of Chan and Roddick (2005)

## 2.1 Mobile Wireless Technologies

In contrast to most distributed networks, the nature of the communication links used in mobile environments determines to a great extent the appropriate protocols that are required to be adopted for data management. Therefore, this section briefly describe the three major classes of wireless technology based on their operating range.

Wireless Personal Area Network (WPAN) describes wireless networks that have a small transmitting range, commonly less than 10 metres. As a result of this short range, they usually have low power consumption. An example of a WPAN technology is Bluetooth – an open standard for shortwave radio communications that allows two-way connectivity of up to 1 Mbps. The concept of WPAN is based on proximity networking. That is, devices may come and go frequently and devices may join or collaborate with each other or other networks in proximity as the WPAN user moves. Current Bluetooth technology allows any Bluetooth enabled devices to communicate when they are within proximity.

The Wireless Local Area Network (WLAN) is a wireless extension of the traditional Local Area Network (LAN) (Imielinski & Badrinath 1994). Like LANs, they provide medium coverage of up to a few hundred of metres, at relatively high data rates. Common examples for WLAN are the IEEE 802.11b and 802.11g standards that provide data rates of up to 11 Mbps and 54 Mbps respectively. As a result of the medium range and high data rates, energy consumption is relatively high and only portable systems such as laptops that can supply the required energy, while keeping a reasonable battery life, are able to continuously support such networks. WLAN works more efficiently when used in stationary networks. That is, the mobile unit is restricted to the coverage of the wireless access point and roaming between networks is typically difficult.

The Wireless Wide Area Network (WWAN) covers network systems capable of providing services that include voice, data and video, to geographically large areas (kilometres). These networks use a cellular architecture, where an area is divided into cells. Each cell consists, at minimum, of a base transmitting hardware which interacts with both mobile devices in its area of service and other base transmitting hardware from different cells. The first generation of these wireless networks were designed mainly for voice communications and utilised analog technology such as FM modulation. The second and current generation of wireless networks use digital modulations to provide voice and limited data services. This commercially accepted generation of wireless network includes

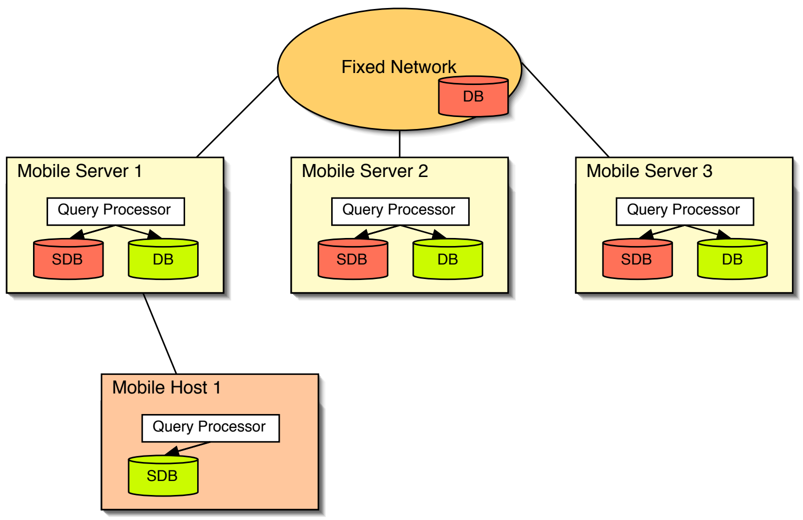


**Figure 2.1. Architecture for a Mobile Database**

systems such as the Global System for Mobile (GSM), Code Division Mutiple Access (CDMA) and paging networks. The third generation of WWAN, or more commonly known as 3G networks, evolves from mature second generation systems and is currently appearing on the commercial market. Its aim is to provide a single set of standards for high voice, data and video communication rates for a wide range of wireless applications.

The above mentioned technologies are terrestrial that employ land-based hardware, such as base towers, to provide wireless communications to mobile computers. However, it is also possible to use satellites to relay the communications. For example, the Iridium System employs 66 low-earth-orbit networked satellites to provide wireless voice and paging coverage to mobile devices (Leopold, Miller & Grubb 1993).

## 2.2 Mobile Database Architecture

A typical architecture for a mobile database includes a small database fragment residing on the mobile device that has been derived from a complete database, (see Figure 2.1 (Madria et al. 1998)).

This architecture may be considered as a client/server architecture where the complete database is located within the server and the summary database resides within the client or mobile device. This type of architecture has been used in the literatures of many projects (Demers, Petersen, Spreitzer, Terry, Theimer & Welch 1994, Phatak & Badrinath 1999, Chan & Roddick 2003, Madria et al. 1998). The Bayou architecture (Demers et al. 1994) uses a client/server architecture with the addition of *lightweight* servers, which are clients capable of serving data in their caches to other clients. Such a technique allows for better availability of data but has a weak consistency property that requires complex conflict detection and resolution.

Other database architectures have also included an agent within the structure, i.e., a server/agent/client architecture. The role of the agent varies between different implementations and resides either in the server or client mobile device. A bandwidth dependent agent, called an *intermediary* is described by Zenel and Duchamp (1995) to allow only essential data to travel through a slow communication link, while other data are filtered out and delayed. In a paper by Heuer and Lubinski (1996) an *information agent* is introduced to intelligently handle the transfer of multimedia specific data. Lauzac and Chrysanthis (1998*a*, 1998*b*), introduced a *view holder* agent to provide individual updates to the client's database. Pitoura and Bhargava (1995*a*), introduced an agent by which mobile transactions can access other heterogeneous database systems.

These client/server network architectures discussed a fixed topology network where the server is fixed and mobile computers may move from one server to another as required. Fife and Gruenwald (2003) propose an *ad-hoc* wireless network system that changes its network topology frequently as the nodes in the area reorganise themselves. These nodes consist of both clients and servers that are both wireless and mobile in comparison to the fixed servers. Such changes in network topology result in the possibility of nodes being in a standalone network or part of a larger network. An example of *ad-hoc* networks is the Bluetooth scatternet, a collection of connected *piconets*, where a piconet comprised of a master node that coordinates all proximate mobile nodes (Law & Siu 2001). This leads to a peer-to-peer architecture in which clients can communicate with other clients to share data. This will result in high data availability but compromises consistency if clients are allowed to perform updates upon any data.

### 2.2.1 Mobile Operation modes

Due to resource restrictions, there are several modes of operation that mobile systems experience when compared to non-mobile systems, which are either fully connected or disconnected. Indeed, many non-mobile systems are written assuming only a fully connected status and are rendered unavailable otherwise. Mobile systems do not normally have this luxury and special protocols are required to handle each mode (Neumann & Maskarinec 1997, Pitoura & Bhargava 1993). The three modes of operation that are of interest here are:

- **Full connection mode**
  The mobile computer is continually connected to a server. Hand-over protocols are required if a cellular structure is used and when mobiles move from one cell to another. The hand-over protocol involves a new communication link between the mobile unit and the new server, requiring the saving and transferring the states from the old to the new server (Madria et al. 1998). This communication hand-over should be transparent to both users and applications not specifically involved with the hand-over process.

- **Disconnected mode**
  In which frequent disconnections of a mobile host, as described in Section 1.2, must be addressed. One possible solution to resolve the disconnection is to use a proxy for the mobile computer (Stanoi, Agrawal, El Abbadi, Phatak & Badrinath 1999). This would ensure the query continues to run even when the mobile component is disconnected. The mobile unit, then, requests an update from its proxy upon reconnection. In addition, mobile computers can voluntarily move into this disconnected mode when idle or low on battery power, to free up bandwidth resources and extend battery life (Pitoura & Bhargava 1993). While disconnected, any applications that required the communication link before the disconnection save their current communication state, and where possible continue with its other processes. Upon reconnection and depending on the saved communication states, applications may resume transmission or reception, or retransmit a request to begin the communication anew.

- **Partial or weak connection mode**
  In which the mobile unit is connected to the rest of the network through low or intermittent bandwidth. The degree to which the communication bandwidth is available varies between less than full bandwidth availability to almost no connectivity. Partial or weak connection occurs when the mobile units are in areas within or on the edge of a cell, where reception is poor. The partial-connection protocol would then be required to enable the mobile client to limit its communications to the available network. Applications using this protocol can then experience longer transmission time and be required to extend timeouts in anticipation of a lengthy response time.

These operation mode protocols could also be deliberately invoked by the mobile unit to allow for resource conservations. Previous research has focussed on the disconnected and partially connected operation modes (Keunning & Popek 1997), since, in these modes, the mobile device must rely more heavily on its own resources to process transactions.

## 2.3 Mobile Database Functionalities

The database management system provides functionalities to ensure the database is operating correctly. This section explores different functionalities such as query processing, replication schemes, concurrency, transaction support, and system recovery.

### 2.3.1 Query Processing and Optimisation

An important issue when dealing with queries is the way in which they are processed. The process of deciding the optimum approach in a mobile distributed environment is difficult as the mobile computers are prone to a dynamically changing environment and state. In previous work by Roddick (1997), three approaches were identified:

- Allow a query processor to determine the appropriate database to answer the query, whether it is the main or summary. This relies on in-built intelligence that can determine whether the local database can answer arbitrary queries.
- Adopt a course grained approach where the query is sent to both databases and the first received response will be used. This option can be expensive but is guaranteed to produce the quickest answer.
- Adopt a fine grained approach in which the query is segmented and those segments are run, either in parallel or in series, on both databases. This has the advantage of utilising the quicker local database more often and, on occasion, can obviate the need for parts of the query to be processed. The parallel option is quicker but results in redundant communication. The serial option uses the complete database when the local database is unable to process the required sub-query.

It is also important to understand the nature of returned responses when using an incomplete database such as a summary database. Madria et al. (1998) identified four types of query answers:

1. Complete[2] and sound[3],
2. Potentially understated (sound but may be incomplete),
3. Potentially overstated (complete but may not be sound),
4. Wrong.

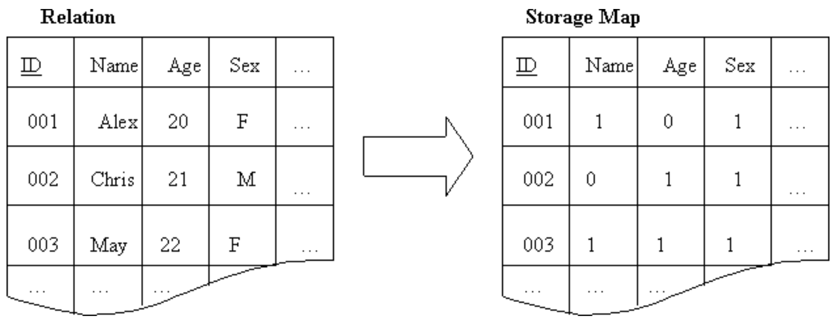
The difference between 1 and 3 lies in the relaxation of the closed world assumption (Reiter 1978), which dictates that anything not recorded within the database may be assumed false, and thus potentially overstated responses are correct if there is no evidence to prove otherwise. This can be useful as part of a fine-grained query decomposition process (Roddick 1997).

With traditional database queries, the objective is to provide query answers that are logically complete and sound. However, summary databases are, by their nature, associated with data loss and are thus sound but incomplete. Consequently, it is not always possible to provide both sound and complete answers. However, for these cases, it is sometimes possible to provide *knowingly overstated* responses (for example, through generalised responses). Thus, query responses can be the same as those given by the original database, or *where allowable*, potentially overstated responses can provided. Furthermore, the query processor is responsible for determining the correct database to answer a query depending upon conditions such as user interaction, heuristics, the nature of the query or type of connections (Madria et al. 1998). Once determined, query rewriting techniques can be used to provide sound if not complete answers to the query. Ganguly and Alonso (1993) presents three search algorithms, namely adapted partial order dynamic programming, linear combinations and linearset algorithm, which are used to determine energy efficient query optimisation within a mobile environment.

This thesis examines the use of Storage Map (SM) within summarised databasase (Chan & Roddick 2003). SM represent the data contained within a summary database in order to answer queries. An SM for a relation is a binary

[2]A query on the summary database will return at least the data that the same query would return on the main database.

[3]A query on the summary database will return no additional data that the same query would return on the main database.



Relation

| ID | Name | Age | Sex | … |
|---|---|---|---|---|
| 001 | Alex | 20 | F | … |
| 002 | Chris | 21 | M | … |
| 003 | May | 22 | F | … |
| … | … | … | | |

Storage Map

| ID | Name | Age | Sex | … |
|---|---|---|---|---|
| 001 | 1 | 0 | 1 | … |
| 002 | 0 | 1 | 1 | … |
| 003 | 1 | 1 | 1 | … |
| … | … | … | | |

**Figure 2.2. Storage Map**

representation of the relation indicating the existence of a value within the summary database, as shown in Figure 2.2. Primary keys are not converted as they are used as references back to the tuples in the actual relation. A fine grained approach is then adopted by using the storage map to determine which data may be extracted to answer the query. Therefore, while disconnected, only the local database is available, an incomplete answer can be returned, but this will gradually be more complete as more of the database becomes available. SM their use are discussed in more details in Chapters 4 and 8.

The concept and semantics of the null value in relational databases has been discussed widely since the introduction of the relational data model in the late 1960s (Codd 1970, Lacroix & Pirotte 1976, Maier 1983, Zaniolo 1984, Roth, Korth & Silberschatz 1989). With the introduction of highly mobile distributed databases, the semantics of the null value needs to expand to reflect a localised lack of information that may not be apparent for the global/complete database. In order to preserve the accepted soundness and completeness criteria, this thesis extends the notion of nulls to include the semantics of 'local' nulls (Chapter 6).

## 2.3.2 Data Replication Schemes

Data replication schemes specify the way in which a replicated site can update the database and how that update is propagated to other sites. There are two mechanisms available that relate to where an update is processed. The first designates a primary site through which all updates are performed and from which updates are propagated. The second uses an 'update everywhere' approach whereby the update is processed at the site of origin and propagated to all sites that have an interest in the data (Wiesmann, Pedone, Schiper, Kemme & Alonso 2000*a*, Wiesmann, Pedone, Schiper, Kemme & Alonso 2000*b*).

Within distributed databases, there are generally two propagation protocols, eager or lazy. An eager update protocol keeps the propagation process as part of the transaction, requiring all sites to be synchronised during an update (Gray, Helland, O'Niel & Shasha 1996). Further classifications may be possible in terms of how the transaction is terminated, such as through voting or non-voting techniques, and the manner in which the servers communicate with each other (that is, constant or linear interaction) (Wiesmann et al. 2000*a*). Eager updates have the advantage of providing serialisability of the update execution on each of the sites and consistency between all the replicas (Breitbart, Komondoor, Rastogi, Seshadri & Silberschatz 1999). However, it requires higher message overheads and extended response times as all the replicas are involved in the update process (Wiesmann et al. 2000*b*). While eager updates work well for traditional distributed databases, it is often unsuitable for mobile distributed databases as frequent disconnections may cause the update to fail.

Lazy update protocols, on the other hand, propagate an update only after it has commit on the initiated site(Wiesmann et al. 2000*b*, Gray et al. 1996, Breitbart et al. 1999, Ladin, Liskov, Shrira & Ghemawat 1992). Lazy updates can provide faster response time, since other replicas are only involved after an update has been committed, but global serialisability may be compromised since it is possible for two or more sites to commit the same data at the same time (Gray et al. 1996). Therefore, further schemes are required to detect and resolve such conflict.

The combination of both eager and lazy schemes is examined by Lubinski and Heuer (2000). The authors discussed a framework for a protocol to allow replication that is configured for specific mobile applications.

There is a large body of literature dealing with the different schemes, most of them comprising various combinations of the classifications discussed above. Many of these are also surveyed in other literatures (Bernstein, Hadzilacos & Goodman 1987, Davidson, Garcia-Molina & Skeen 1985, Wiesmann et al. 2000*a*, Wiesmann et al. 2000*b*).

### 2.3.3 Concurrency Controls

Concurrency control mechanisms are used to ensure consistency within a database management system when multiple users access and change data. Concurrency control methods are usually dependent on where update procedures can occur. For example, in cases where updates only occur on the central or primary site, simple two-phase locking management on that site provides the required consistency. For a distributed database system, where updates occur at any site, a voting method is used in which the site performing the update requests a lock from all relevant replicas (Elmasri & Navathe 2000), locking the data for the update duration. In mobile distributed database systems, however, such locking procedure is costly in terms of an increase in the number of messages that are required to coordinate the locking and unlocking of data while the mobile site is moving from one server to another (Jing, Bukhres & Elmagarmid 1995). An Optimistic Two Phase Locking for Mobile Transactions (O2PL-MT) was proposed by Jing et al. (1995) to reduce the number of messages, based on the Optimistic Two Phase Locking (O2PL) algorithm presented by Carey and Livny (1991). O2PL follows an optimistic 'read one, write all' concurrency control approach, which allows lock processing for read locks on the site or the nearest server site. O2PL-MT then restricts the read unlock messages to the remote server where the lock is set by allowing unlocks to occur at the local or nearest server site.

The Bayou storage system (Terry, Theimer, Petersen, Demers, Spreitzer & Hauser 1995), provides a weakly consistent system that ensures eventual consistency between replicas. The Bayou architecture achieves high data availability to mobile applications at the cost of providing weak consistency, by allowing any client or server to perform updates (Demers et al. 1994). Conflict detection and resolution for the Bayou system is possible through the use of dependency checks and merge procedures, but is mainly application specific. That is, each write operation has additional queries attached that checks the server's data against an expected result to specify if a conflict has occurred. If a conflict occurs, a merge procedure will attempt to resolve it.

The authors (Pitoura & Bhargava 1995*b*, Pitoura 1996) proposed a two-level consistency model where clusters are formed from different partitions of a database system. Within each cluster, full mutual consistency is maintained, while only varying degrees of consistency are kept between clusters. This is enabled through the proposal of strict and weak operations, where strict operations require full consistency, while the weak operations provide solutions that may be weakly consistent. Assuming that partitions are specified through usage patterns, allowing varying consistencies between clusters are acceptable, as the main users for any particular data would be within a cluster.

### 2.3.4 Transaction Support

Transaction support relies on the availability of all necessary information to complete its operations. If data are unavailable, the transaction would either fail or be required to wait for the data to be available.

Traditional transactions are a sequence of read and write operations that include operations that access different data over multiple locations if a distributed database is used. Mobile transactions however, are different and display different characteristics to both centralised and distributed systems. These characteristics include (Madria 1998, Dunham & Kumar 1999)

- the need to split a transaction into parts for computation on both local and remote databases,
- migration of the transaction to a remote non-mobile database when no user interaction is required to conserve the mobile computer's resources,
- support for the transaction by non-mobile databases,
- computation of different parts of a transaction on different servers while the mobile computer is on the move, and
- a long-lived nature due to the frequent disconnections that may occur.

Chrysanthis (1993) proposed an open-nested transaction model using the notion of *Reporting Transactions* and *Co-Transactions* in order to perform updates on mobile computers. A mobile transaction may be split into several subtransactions which may commit and abort independently. A reporting transaction allows partial results to be shared with other subtransactions in the mobile transaction, while a co-transaction allows the control of the partial results to be passed from one subtransaction to another at the time of sharing.

For multidatabase systems with mobile components, Yeo and Zaslavsky (1994), presented a method that allows multidatabase transactions to be submitted by a mobile computer. Once submitted, a mobile computer may disconnect and a coordinating site will process the transaction on its behalf. Using simple messaging and queuing techniques, the mobile computer may be informed of the status of the transaction when it reconnects. Such technique is similar to having a proxy for the mobile computer that process transaction requested by the mobile computer.

A semantic-based transaction processing method has been extended for mobile transactions by Walborn and Chrysanthis (1995), who introduced the idea of fragmentable and reorderable objects to improve database concurrency and caching efficiencies. Using this idea, a mobile computer requests a database fragment which is cached locally, and on which mobile transactions operate. This fragment is consistent as it is only accessible to the transactions of the mobile computer that requested it. Once a fragment is no longer required, it is merged back into the master database. Such a technique is suitable when data objects can be fragmented such as aggregate items, sets and stacks.

*Kangaroo Transactions* were introduced by Dunham, Helal and Balakrishnan (1997) to allow transactions to be processed by capturing the moving nature of a mobile computer. In effect, a transaction is split into several parts whereby each part is processed (and perhaps committed) by the server in which the mobile computer relies. These parts or subtransactions are serialisable using the split transaction method proposed by Pu, Kaiser and Hutchinson (1988), and may commit and abort independently.

Madria et al. (2002) examined a multi-version transaction model for mobile computing. The model aimed to increase the availability of data by using versions. By allowing a version of a transaction to be executed and committed on the mobile computer, the new data from that transaction would be readily available. A *terminating* version is then required at the main database to ensure the consistency of the database.

### 2.3.5 System Recovery

In centralised and primary copy distributed database systems, the failure of the primary copy prevents clients from putting in an update request. In instances where clients do not cache any data locally, read access would also be denied. In cases where the failure duration is extensive, a new site must be nominated to fill in the role as the primary copy database. The most common algorithm for deciding is an election algorithm proposed by Garcia-Molina (1982). The algorithm specifies a site nominating itself as primary and informing the remaining sites. Once a majority agreement is reached, the site becomes the new coordinator until the actual primary coordinator has recovered. Similar algorithms may be used in mobile distributed systems with fixed sites, where a fixed site may assume the role of primary coordinator. It is unlikely for a mobile computer to take on the role of primary coordinator since they may disconnect frequently from the network.

In mobile ad-hoc networks, network partitioning may occur when mobile computers move from one area to another. Network partitioning occurs when a network is split into different partitions in such a way that communication between partitioned networks is not possible. The authors, Aggarwal, Kapoor, Ramachandran and Sarkar (2000), introduced a clustering algorithm, where the elected super master knows all the mobile computers that are available, to design and create a scatternet.

## 2.4 Summary

For databases to operate within the mobile environment, the adoption of different protocols that suit these environments are required. Current protocols that work well for traditional distributed databases are, however, unsuitable for mobile environments. They must therefore, extend their functionalities to handle the limited resources exhibited by the mobile computers.

This chapter introduced the area of mobile computing and databases. In particular, it examined the different wireless technologies enabling a database to be mobile and the current mobile database architectures. Additionally, a survey was provided regarding issues resulting from the effects of mobility on different database functionalities and the research undertaken to resolve them.

# Chapter 3

# A Survey of Data Summarisation

To date, many database summarisation techniques used on traditional databases involve the use of structural reduction techniques that reduce the volume of data without considering the use (i.e., the importance to the user) of the data. However, in mobile distributed environments context sensitive information is important in as it provides more relevant information to users readily and efficiently (Hawick & James 2003, Jones & Brown 2004, Covington, Long, Srinivasan, Dev, Ahamad & Abowd 2001, Chan & Roddick 2003). Context sensitive data are the part of information that has a semantic connection to user of those data.

Database summarisation involves the reduction of the size and information capacity of a database while maximising the useability of the resultant (summarised) dataset (Roddick, Mohania & Madria 1999). That is, the measure of a summarisation technique can be seen as the relationship between the physical reduction in dataset size and the loss of *useful* information as a result. Clearly, this relationship is non-linear as some items in a dataset will be more critical than others with respect to context. In simple terms the effectiveness of a summarisation technique can be given as:

$$W = \frac{\zeta_{DB}}{\zeta_{SDB}}.\frac{\Upsilon_{SDB}}{\Upsilon_{DB}} \tag{3.1}$$

where $\zeta_{SDB}$ and $\zeta_{DB}$ are the *storage requirements* and $\Upsilon_{SDB}$ and $\Upsilon_{DB}$ are the *useful information capacities* of the summarised and original databases respectively (Roddick et al. 1999). Given that $\zeta_{DB}$ and $\Upsilon_{DB}$ are fixed for any given database and $\zeta_{SDB}$ has an upper limit on any given platform, the task is to maximise $\Upsilon_{SDB}$ for a target system.

[1]A version of this chapter appears as part of Chan and Roddick (2005)

This chapter explores both context sensitive and insensitive techniques used for traditional, distributed and mobile databases. The chapter is structured as follows: Sections 3.1 and 3.2 discusses techniques that modifies the schema and domain of a database, respectively. Section 3.3 presents the techniques used to allocate data to different databases within a distributed database system. Section 3.4 examines data compression techniques. Finally, Section 3.5 provides a discussion for this chapter.

## 3.1 Schema Modification Techniques

These techniques modify the schema by removing either attributes or tuples from the relations of a database to produce new relations that are reduced in size. Schema modification techniques by themselves are usually insensitive to data usage. However, the inclusion of usage sensitive criteria such as a previous usage history, to provide context sensitive techniques. There are three techniques presented here as schema modification techniques.

### 3.1.1 Projection (vertical) methods

Simply used, this method involves the vertical elimination of attributes in a relation within the database (Ceri, Negri & Pelagatti 1982, Lubinski 2000, Roddick et al. 1999), and is a common technique used for reducing the size of a database. As shown by Roddick et al. (1999) the summarisation weighting W for each attribute is determined by:

$$W(t) = \frac{\zeta_{DB}}{\zeta_{SDB}} . \frac{K_{DB}}{K(\pi_t(DB))} . \frac{\Upsilon_{SDB}}{\Upsilon_{DB}} \tag{3.2}$$

where

$t$ is an attribute,

$DB$ is the complete relation,

$SDB$ $(= DB - t)$ is the summary relation,

$\zeta$ is the storage requirement,

$K$ is the cardinality of a relation, and

$\Upsilon$ is the information capacity of a relation.

The method of projection may also be utilised in conjunction with other types of summarisation techniques, such as concept hierarchies. Projection provides a modification to the structure of the database schema. That is, the technique is used to remove attributes within the base relations.

### 3.1.2 Selection (horizontal) methods

Selection involves the horizontal reduction of tuples within a database (Chu 1992, Cornell & Yu 1990, Muthuraj, Chakravarthy, Varadarajan & Navathe 1993, Navathe, Ceri, Wiederhold & Dou 1984, Navathe & Ra 1989, Lubinski 2000, Roddick et al. 1999). This is achieved through a selection of values of a specified attribute. The summarisation weighting W is determined as follows (Roddick et al. 1999):

$$W(c) = \frac{K_{DB}}{K(\sigma_c(DB))}.\frac{\Upsilon(\sigma_c(DB))}{\Upsilon_{DB}} \tag{3.3}$$

where

$c$ is the selection criterion,
$DB$ is the complete relation,
$\sigma_c(DB)$ $(= SDB)$ is the summary relation,
$K$ is the cardinality of a relation, and
$\Upsilon$ is the information capacity

Again, selection may also be found in conjunction with other summarisation techniques.

### 3.1.3 Hybrid methods

Projection and selection can be combined to produce hybrid method or views. Views define a function from a subset of relations to a derived relation, and are *materialised* by physically storing the tuples of the view (Lauzac & Chrysanthis 1998*b*). The use of views in terms of mobile systems has largely been discussed within data warehouse applications to allow maintenance and updates of the mobile client's database (Lauzac & Chrysanthis 1998*a*, Lauzac & Chrysanthis 1998*b*, Stanoi et al. 1999). Lauzac and Chrysanthis' *materialised view holder*, acts as a proxy during network disconnection to provide the required updates and views to the mobile client upon its reconnection. Note that since views are predetermined queries, an equivalent query employing rewriting techniques can also be used to allow query optimisation (Halevy 2001).

A related method used in distributed databases is that of hybrid fragmentation, of which there are different schemes. For example, if horizontal fragmentation is performed followed by vertical fragmentation then subsequent rejoining of the fragments (assuming each fragment is disjoint) vertically, will produce the horizontal fragment (See Figure 3.1). Horizontal joins, however, will not be possible since the fragments may not contain the same attributes. The same occurs where vertical fragmentation is performed before horizontal fragmentation, with the exception that vertical joins are not possible and horizontal joins will produce the original vertical fragment (See Figure 3.1). The hybrid fragmentation method is independent of the sequence that the fragmentation is performed (Navathe, Karlapalem & Ra 1996). The fragments, termed *grid cells*, have the property of being able to be both vertically or horizontally joined. This is possible since each grid cell belongs to exactly one horizontal and one vertical fragment (See Figure 3.1).

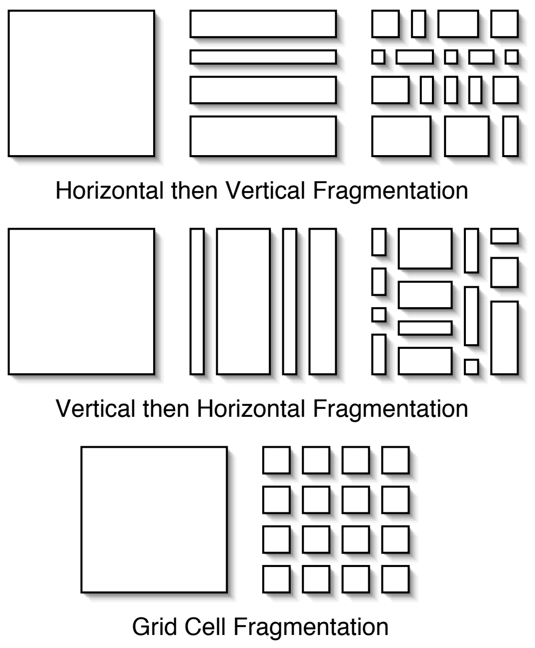

Horizontal then Vertical Fragmentation
Vertical then Horizontal Fragmentation
Grid Cell Fragmentation

**Figure 3.1. Hybrid Fragmentation Methods**

## 3.2 Domain Modification Techniques

Domain modification changes the domain of a relation in such a way that it is possible for data to be grouped together or replaced with associated data. This then produces a reduction in size of the final result. Domain modification techniques discussed in this section include concept hierarchies, abstraction and surrogacy.

### 3.2.1 Concept Hierarchies

The usage of concept hierarchies to reduce the database size is a domain modification technique. This technique was proposed by Madria *et al.* (1998) as a query processing model for mobile computing to facilitate data volume reduction for a summary database. The concept hierarchy process is capable of organising data and concepts in a hierarchical form by providing a mapping or generalisation of the lower layer concepts to their corresponding higher level concepts (Han & Fu 1994). The central idea is that of concept ascension in which low level data items are elevated onto a higher level within the hierarchy and duplicate tuples are then removed. This hierarchy may be externally supplied, or dynamically generated and refined, as demonstrated by Han and Fu (1994). In terms of context sensitivity, current techniques for concept hierarchies may be considered insensitive to the data usage since they do not have the capability of identifying specific data for storage. Thus, while still a useful addition to schema decomposition, these techniques can not be determine the optimum data required by the user.

### 3.2.2 Abstraction through aggregation

This is the summarisation of selected attributes through the use of metadata by creating new attribute values such as the sums and averages of existing data (Heuer & Lubinski 1998, Lubinski 2000).

A report by Barbara et al. (1997) examined several other aggregation techniques that may provide rapid but approximate answers for data warehouse applications. These aggregation techniques include Singular Value Decomposition (SVD), Discreet Wavelet Transform (DWT), Regression, Histogram, Clustering Techniques, Index Trees, and Sampling techniques.

SVD technique decomposes a matrix, $N \times M$ for example, into three matrices, $N \times r$, $r \times r$ and $r \times M$. That is, (Press, Teukolsky, Vetterline & Flannery 1996),

$$X = U \times \Lambda \times V^t \tag{3.4}$$

where $U$ and $V$ are then similarity matrices for the row and column, respectively, of $X$, and $\Lambda$ is an diagonal $r \times r$ matrix. These matrices are then used to determine which group the data will fall under. The SVD technique and its properties are discussed in more details by Faloutsos (1996) and Korn et al. (1997).

DWT is a signal processing technique used to determine the groups that data in a dataset may fall into. There are several wavelet transformation techniques available including the *Haar* transform (Barbara, DuMouchel, Faloutsos, Haas, Hellerstein, Ioannidis, Jagadish, Johnson, Ng, Poosala, Ross & Sevcik 1997), Daubechies-4 and Daubechies-6 transforms (Daubechies 1992). For the datasets, a *k-d signal* used in wavelet transforms represents a k-dimensional matrix. For all wavelet transformation techniques, two functions are applied to a signal, for example, a 1-d signal. The first function performs a weighted average while the second performs a weighted difference. The functions are then recursively applied to both halves and sub-halves of the signal as indicated by the weighted average and differences. This continues until the resulting signals are too short to transform.

Regression attempts to model the data as a function of the values of a multi-dimensional vector. Linear regression is the simplest form of this whereby a variable $Y$ is modeled as a linear function of a another variable $X$ (Wonnacott & Wonnacott 1985). That is, $Y = \alpha + \beta X$. The parameters $\beta = \frac{\sum(X-\bar{X})(Y-\bar{Y})}{\sum(X-\bar{X})^2}$ and $\alpha = \bar{Y} - \beta\bar{X}$ are estimated using the current known data $X_1, X_2, ..., Y_1, Y_2, ...$ and $\bar{X}$ and $\bar{Y}$ are average values for those data respectively.

Histogram is a common form of aggregation technique(Poosala, Ioannidis, Haas & Shekita 1996), which works by distributing data into *buckets*. *Buckets* may be considered to be subset in which data may fall into. Approximate answers to the true attribute value and their frequencies in the data may then be provided using the history stored with each *bucket*. The histogram technique is mainly used to record data statistics for query optimisers.

Clustering is used to determine groups within a dataset (Kaufman & Rousseeuw 1990). These clusters consist of a collection of similar data objects. Similarities are usually determined using a distance function. By using the derived clusters, it is then possible to approximate answers by searching the clusters instead of the actual data.

Index trees are used for organising and accessing large datasets. Since the nodes of an index tree represent the aggregate information of a data set, it is possible to use these nodes as a form of aggregation for data reduction (Antoshenkov 1993, Aoki 1998). Index trees are usually ordered on one or more key attributes. The B+− is commonly used in one-dimensional data sets where there are only single key attributes (Bayer & McCreight 1972, Comer 1979). The single keys are then ordered according to the different ranges to provide nodes in a tree. R-trees on the other hand, deal with higher dimension data sets where more than one key attribute is used for (Guttman 1984). Thus, regions which the data are organised into, forms the nodes of a R-tree.

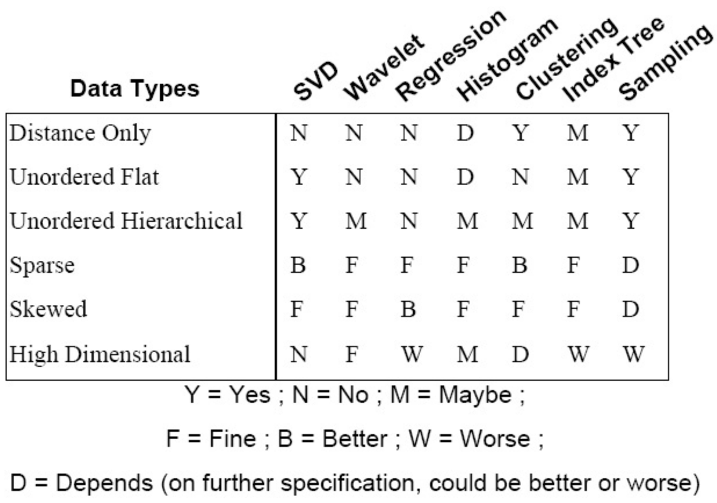
Sampling attempts to represent a large data set with a small random sample of data elements. Many techniques for sampling data exists, and most of which are dependent on the application of the data set (Cochran 1977, Sudman 1976, Sarndal, Swensson & Wretman 1992). Two of the more common types of sampling techniques are the simple and cluster techniques. The simple random sampling technique attempts to extract a sample $n$ records from a set of $N$ records such that $N >> n$. Whiet in cluster sampling, the records are first clustered into mutually disjoint clusters and then a sample is made for each cluster.

Given the different types of aggregation techniques available, there are also different types of data sets to which the reduction techniques are more suited to. These include distance only, unordered flat and hierachical, sparse, skewed and high dimensional data types. Figure 3.2, extracted from Barbara et al. (1997), shows the data types that are applicable for each aggregation technique.

Aggregation, allows not only a reduction in data required for storage by generalising the data into a new collection of objects, but is capable of quickly answering queries through those new objects. All data reduction ultimately result in some information loss where an approximate form must be used to replace that data. Therefore, aggregation works well if the user does not require specific or detailed data.

### 3.2.3 Surrogate

Finally, in situations where complex data types are used in a database, a surrogate method can be used. Data that are complex can usually lead to extra resources



| Data Types | SVD | Wavelet | Regression | Histogram | Clustering | Index Tree | Sampling |
|---|---|---|---|---|---|---|---|
| Distance Only | N | N | N | D | Y | M | Y |
| Unordered Flat | Y | N | N | D | N | M | Y |
| Unordered Hierarchical | Y | M | N | M | M | M | Y |
| Sparse | B | F | F | F | B | F | D |
| Skewed | F | F | B | F | F | F | D |
| High Dimensional | N | F | W | M | D | W | W |

Y = Yes ; N = No ; M = Maybe ;

F = Fine ; B = Better ; W = Worse ;

D = Depends (on further specification, could be better or worse)

**Figure 3.2. Applicability of Techniques to Different Data Types**

being required to access, transfer and present them (Lubinski 2000). Thus, using surrogates that substitutes complex data types with corresponding simpler forms relieves resources for other operations. An example of this is the substitution for a large image file with its equivalent but simpler text description. Such a technique would modify and replace the data type of the attributes of the base relations.

## 3.3 Data Allocation of Fragments

In many literatures, fragmentation and data allocation are considered separate processes. Methods of fragmentation use a representative set of transactions or queries as a base to partition a global relation into fragments. For example, attribute and predicate usage matrices are used as starting points for vertical and horizontal fragmentation procedures, respectively. These matrices are formed by examining the frequency that an attribute or predicate is accessed by the given set of transactions. While basing the fragmentation on a given set of transactions is important, it may not encapsulate the importance of a query, especially when the representative set is small or out of date. Moreover, it can be blunt in its decisions for inclusion. Information, such as location, the time of events stored in the database and inductive association between data items is useful when deciding on the content of a summarised distributed database.

Regarding data allocation, literatures indicate the common usage of a representative transaction set to minimise the cost of accessing and processing queries at each site, and allocates fragments accordingly. This is perhaps the same set as used in the fragmentation stage. Shephard *et al.* (1995) investigate a two-phase approach to data allocation. The first step creates the fragment clusters based upon the given set of transactions and their access frequencies, the second step allocates the clusters based on a divide and conquer algorithm such that the cost of query processing is minimised. This allocation technique is most useful when future access patterns are the same. Ahmad *et al.* (2002) suggest that evolutionary techniques, such as genetic algorithms, simulated evolution, mean field annealing and random neighbourhood search algorithms, can be useful in allocating fragments to each site. These algorithms are also based on a representative set of queries to minimize access costs. While these algorithms are used within traditional distributed database systems, it is also possible to use similar algorithms for mobile distributed database systems in which the access costs also consider the resource limitations of a mobile computer (Huang, Sistla & Wolfson 1994).

Wolfson and Jajodia (1995) proposed a reallocation of fragments during run-time using data collected from recent transactions. Similar work was undertaken by Brunstrom *et al.* (1995) who showed that dynamic reallocation of fragments is significantly improve the performance of distributed databases with changing workloads. In addition, there is also research that examines fragmentation and allocation within a single process. Note that in some instances, elements of replicated fragments may exist (and potentially be updated) at more than one site.

The allocation method that will be discussed in Chapter 5, uses different context sensitive criteria, including location, time, inductive and enumerated criteria. When combined these criteria specify which data are to be included in the mobile database (Chan & Roddick 2003). Similarly, changes in the user's context would induce a change to the data within the mobile database.

## 3.4 Data Compression

There are many methods of data compression based mainly on character encoding and repetitive string matching (Graefe & Shapiro 1991). Techniques that allow the compression of data usually aim to reduce the redundancy that may be found in stored or communicated data (Lelewer & Hirschberg 1987). More details of the available compression techniques may be found in a number of surveys (Bell, Witten & Cleary 1989, Lelewer & Hirschberg 1987, Cannane & Williams 2001). In the context of databases, there are many literatures available that investigate the use of text only data compression techniques(Severance 1983, Cannane, Williams & Zobel 1999, Roth & Van Horn 1993, Cormack 1985, Graefe & Shapiro 1991, Ray, Haritsa & Seshadri 1995). However, one particular method that does consider the different data types is the RAY algorithm as described by Cannane, Williams and Zobel (1999).

## 3.5 Discussion

This chapter discusses data summarisation techniques used within databases. Many of the techniques surveyed are capable of reducing a large volume of the data. However, most of techniques do not consider context or data usage. For mobile environments, using the context of the data may provide the user with more efficient and relevant answers. Thus, in Chapter 4 a discussion on useful context is presented.

While data reduction or summarisation and data compression is similar in that they both reduce the size of a database, they are viewed quite differently within this thesis. In data compression, the aim is to reduce the redundancy in a data, while data summarisation aims to reduce data by removing irrelevant data or aggregating the data together.

# Chapter 4

# The Uses of Context in Mobile Computing

The term context is difficult to define, as recognised by Dourish (2004). It can be defined as being aware of the changes that are occurring to the context. A common definition of context as described by Dey, Abowd and Salber (2001) is "typically the location, identity and state of people, group and computational and physical objects". However, there is more to context than just the objective features, which can be tracked and recorded easily. There are also user experiences which may include subjectively perceived features and how past experiences of similar contexts can affect current activities (Chalmers 2004).

In mobile environments, the use of context sensitive, or context awareness is of more importance as mobile computers are essential part of a network whose topology can change constantly (Schmidt 2000, Hawick & James 2003, Imielinski & Badrinath 1992, Zaslavsky 2004). To quote Moran and Dourish (2001):

> "'One goal of context-aware computing is to acquire and utilize information about the context of a device to provide services that are appropriate to the particular people, place, time, events and so forth."'

Moran and Dourish, thus, suggest that noticing the context that a technology is in, is an important step towards building usable technologies.

This chapter is structured as follows. Section 4.1 discusses location awareness as a context sensitive technique. Sections 4.2, 4.3 and 4.4 examines the application of context in information retrieval, data reduction and sensors, respectively.

Section 4.5 explores the characteristics of context. Finally, Section 4.6 provides a summary of this chapter.

## 4.1 Location-aware techniques

In regards to computer mobility, context awareness in terms of location awareness covers a significant portion of the current research. That is, in highly mobile environments, mobile computers have the ability to constantly change their location. Thus, applications capable of adapting to their current location can provide its user with more useful information.

Early location-aware research focus upon ubiquitous computing techniques and from which general consideration of the use of context in mobile computing began (Schilit, Adams & Want 1994). Other research lead to the development of the Active Badge system, which requires a device to transmit infra-red signals every intervals so that networked sensors may determine the location of that device (Want, Hopper, Falcao & Gibbons 1992, Ward, Jones & Hopper 1997). Many systems that exploit location awareness information are based on describing the user's physical location. These information are easily collected using Global Positioning System (GPS) or location information available from the underlying communication infrastructure, such as GSM.

While it is useful to have location only context, the literature also explore the combination of location and time to capture context-aware information. Brown (1996) developed Stick-e-notes, which are documents that are tagged with both location and time information. This eventually led to the development of an electronic tour guide application, which provided tourist information based on their position and orientation (Brown, Bovey & Chen 1997).

Hawick and James (2003) use the contextual information of both spatial temporal information within a middleware architecture. Furthermore, the authors introduce an active list of preferences that combines a list of user preferences, spatial and temporal information. The middleware supports mobile applications by attempting to filter the data according to the active preferences. For example, an email destined for the user, first passes through the middleware. It then determines (depending on the user's current location, the current time and any other preferences) whether the email should be sent to the mobile application.

Location-awareness has also been incorporated through modifying the query language. That is, a query in SQL is modified to include the location of the user

(Imielinski & Badrinath 1992). So for example, if a user queries a database using *Find me a doctor*, this is then be modified into *Find me a doctor that is less than 5km away from me.* In terms of SQL, we would then have:

```
SELECT  Name   FROM Doctor
```

Assuming that the system determines that my location is $l_1$, and the location within 5km of $l_1$ are $l_2$, $l_3$, $l_4$, then the following select statement is possible:

```
SELECT  Name   FROM Doctor
        WHERE  location = 'l1'
        OR     location = 'l2'
        OR     location = 'l3'
        OR     location = 'l4'
```

Location is one of the more commonly used pieces contextual information within mobile environments. Its use in middleware and query modifications shows its capablity of reducing the data to be presented on the user's mobile device.

## 4.2 Information Retrieval

Information retrieval is not only limited to databases. The tour guide application mentioned earlier is an example of an information retrieval system. The use of location-aware context allows the electronic tour guide, Cyberguide, to provide information to a tourist based on the knowledge of position and orientation (Abowd, Atkeson, Hong, Long, Kooper & Pinkerton 1997). The *Guide* Project developed in Lancaster, attempts to integrate both context of location and user preferences in its information retrieval system. This is done to provide an awareness of the service quality to its users, while employing a wireless network architecture.

Another example of an information retrieval system is that of a web search engine. Given a search condition, the retrieval engine attempts to return a set of potentially relevant documents. This is done by giving a weight to each document according to how well they match the query (Jones & Brown 2004). The use of context within information retrieval then determines which of the documents are more relevant to the user, by the use of contextual criteria.

The Jones and Brown (2004) describe two methods, *context diary* and *context-aware cache* to deal with dynamically changing context and fast retrieval of information, respectively. The *context diary* incorporates both the current and predicted location of the user and the user's personalisation to determine which documents are of relevance. The *context-aware cache* attempts to predict the future context that the users may find themselves in. Documents relating to such context will then be retrieved and stored in the cache for fast access.

## 4.3 Data Reduction

Lubinski (2000) proposes the gradual data reduction technique as a way of reducing the data of query results before it reaches the mobile system. This technique specifies different domains of data precision and also a layered reduction for those domains. The domains of data precision relate to the interestingness that the user has placed on the resultant data through specifying domain boundaries. For example, in terms of a mobile user, three domains of data precision can be specified. *Domain 1* can corresponds to data that are most relevant with respect to the user's current position. *Domain 2* contains data that are less relevant to the user's current position but could be relevant to future positions. Finally, *Domain 3* contains data that are not relevant to any of the user's positions.

The layered data reduction technique, where at *Layer 1* the least or no data reduction occurs, up to *Layer n*, where the most extreme data reductions occur, are used to reduce the data contained within those domains from 1 to *n*. That is, for *Domain 1*, a corresponding *Layer 1* will be used. So following the previous example, an abstraction technique may be used where certain parameters are changed to allow weak to strong abstraction to be used. *Domain 1*, *Layer 1* can be defined where no reductions are performed. While *Layer 2* where weak abstraction is performed to the data in *Domain 2*. Finally, a strong abstraction technique is used in *Layer 3* for data in Domain 3.

## 4.4 Sensors

The use of sensors provides a means to collect context information in addition to that of location. Sensor technology is widely used in applications involving robotics, machine vision and process control. There are many types of sensors available including optical/vision, audio, motion, location and bio-sensors.

For example, the development of motion sensors integrated with a handheld device allows the interface to be aware of gestures made by users (Harrison, Fishkin, Gujar, Mochon & Want 1998, Rekimoto 1996). The idea of wearable computers furthers the development of sensors in context awareness. As such, it has been explored by Maguire, Smith and Beadle (1998) and Smith (1999), where the distinguishing features of these wearable computers are the awareness of both the user and the physical environment.

However, it should be noted that the context obtainable by sensors are generally of low abstraction, such as noise level, temperature, or it must be designed to be usable for specific applications.

## 4.5 Characteristics of Context

The two major characteristics of context are that of objectivity and subjectivity. For computer systems, objectivity refers to context that is not influenced by the system's users. Examples of objective context include location and temporal awareness and were discussed in the previous section. Subjective context on the other hand, are be influenced by the system's users. Examples of subjective context include user's activity history and direct input by the user.

Objective context is usually easier to identify and record than subjective context. However, the need to effectively combine both types of context was recognised by Chalmers (2004). Several authors discuss an activity-based computing system that combines the use of objective context such as location awareness with subjective context, namely modelling of activity using workflow and activity theory as described by Nardi (1996). The activities here are typically a set of defined tasks from which the users may use the computer services available from stationary and mobile computers (Christensen & Bardram 2002, Bardram, Kjaer & Pedersen 2003).

## 4.6 Summary

There is a variety of context information available ranging from the well researched location-awareness context to the subjective context such as user's activities. While this chapter surveys the use of objective context within computer systems, it is also important to consider subjective context. By combining the use of both types of context gathering, it is possible to provide users with more useful information.

The next chapter discusses the COSMOS technique to provide a framework to integrating different types of context in summarising a database.

# Chapter 5

# COntext Sensitive MObile Summarisation (COSMOS)

In this chapter, several context to extract information about the users are examined to provide a solution to determine the storage of data. In general, the selective determination of information to be stored may still require a large amount of storage that is not available to mobile devices. As a result summarisation is required to fit the required information into the mobile device such that it still useful information. Chapter 3 showed that there are many types of summarisation. This chapter presents several types of context to extract information regarding the users to provide a solution to determine the required data for storage. The data selected is stored in a summary database and represented as a Storage Map (SM), which is an abstract view of the a database for the mobile device.

This chapter presents the COSMOS technique and is structured as follows: Section 5.1 examines the COSMOS database architecture. Section 5.2 presents the first stage of creating a COSMOS database. This stage involves the use of priorities to determine the candidates for inclusion into the summarisation process to identification of the relevance of data with respect to the users. The second stage is then presented in Section 5.3, which employs heuristic techniques to construct the summary database.

[1]A version of this chapter appears as part of Chan and Roddick (2003)

## 5.1 COSMOS Database Architecture

In this research it is assumed that COSMOS is used with a primary or centralised database, to which updates are directed and from which data is replicated to other sites. That is, all updates must be directed to the primary database for propagation. This reduces the complexity of monitoring consistency between every database within the architecture as changes to any database will be resolved at the central database that stores the primary copy of that data.

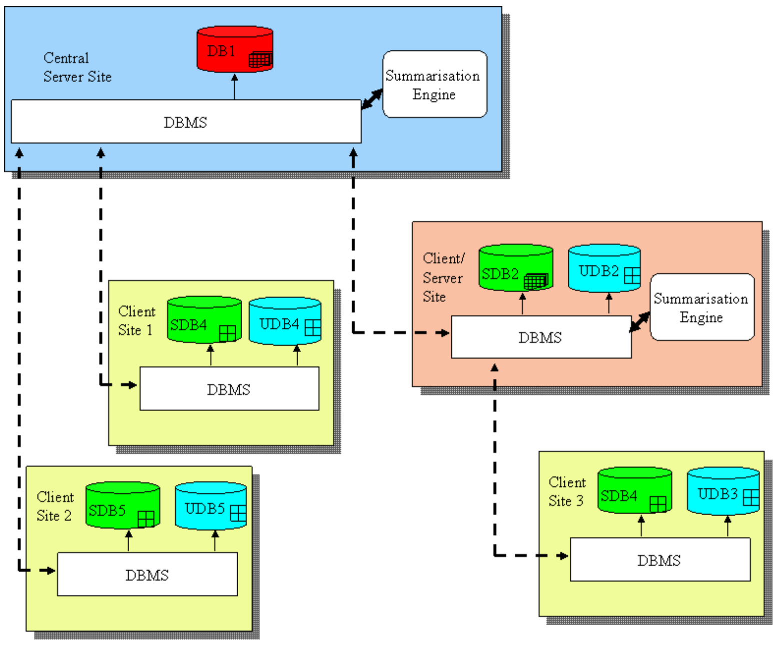


**Figure 5.1. Architecture for a COSMOS Database**

The architecture of a COSMOS database system follows a hierarchical client/ server architecture with its root being a central server database, see Figure 5.1. The hierarchical architecture will consider a database site a server if there exists another site with a subset of its database. A database site is considered a a client when its database is a subset of another site's database. A database site can also be both a client and server if its database is both a subset of another database and a superset for other databases. Being a server requires the site to store additional information of its clients, such as the client's storage maps. This kind of architecture may be useful in situation where, for example, a specialist located at the Flinders Medical Centre Cardiac Unit holds a specialised local database that is a subset of the Flinders Medical Centre's database, which is in turn a subset of the South Australian State Health database.

For the client site, the Database Management System (DBMS) will have fast access to the site's summarised database while maintaining a comparatively slower and perhaps unreliable access to the main database through the server site's DBMS. In addition, each client site's DBMS now has access to an Update Database (UDB). The UDB is a relatively small repository for the changes to the local database that passes through the site's DBMS. The UDB is discussed further in Section 5.1.1.

In the case where the UDB stores data that supersedes the data in the Summary Database (SDB), the DBMS will now be required to adhere to the following steps:

1. The DBMS is required to retrieve data from the UDB by comparing the UDB's storage, $M_u$ may with the query's equivalent storage map, $M_q$.
2. The DBMS then retrieves the remainder of the query from the SDB, using similar procedures as the previous step.
3. The DBMS then attempts to retrieve data unavailable to both the SDB and UDB, from the server.

Thus when data items are available in both the SDB and the UDB, the UDB's data will always be used. This ensures that more updated data found in the UDB is accessed before the local database. If data is not found in the UDB, the main or remote database will be queried. This architecture provides better scalability, in that the main database can be a subset of a larger database, while the local database can also be a superset of other databases, as shown in Figure 5.1.

### 5.1.1 Update Database

A fundamental inclusion to the database architecture used within COSMOS, shown in Figure 5.1, is that of the Update Database (UDB). The presence of an UDB within a local database structure allows a SDB, which was previously specified as read-only, to perform read and write procedures. This provides both the read-only and read-and-write aspect within the COSMOS database architecture. It is important since in cases such as mobile databases, where resources are limited, there are situations where only read-only access is required. In these situations, the mobile database will consists of only the SDB and any updates will require propagation to a server.

Therefore, the UDB can be considered an extension of the SDB and contains more data than is originally assigned to the SDB. The maximum size of both databases is set by the database administrator depending on the available capacity.

The main function of an UDB is to store updated values. Whenever an update is requested, the new values in the UDB are recorded, while rendering the corresponding values in the SDB invisible (through its storage map). A separate UDB storage map is subsequently stored to keep track of which attribute values have been updated. The changed values are stored on the UDB until a database refresh command is initiated. After updating, the UDB's storage map is also updated to reflect the change.

A full or partial pull database refresh integrates new values into the SDB from the central server. Once a full database refresh is completed, the UDB items are no longer required.

## 5.2 A Protocol for Priority-based Data Summarisation (First Stage)

The proposed priority-based data summarisation uses a two stage protocol. The first stage uses the notion of *priorities* (*weights*) that are assigned to every attribute value to indicate the importance of that value. The second stage then selects the appropriate data for the expected situation. To ensure an optimal accommodation of the required information, this is done in a fine-grained manner and are calculated at the level of the data item. The priorities are numerical values, either supplied or generated through observation and experimentation, which are assigned through a multifaceted evaluation of different criteria. The relative use of each criterion is termed a *protocol*, (for example, see Table 5.2). The criteria that might contribute to a protocol are grouped into three categories, Primary, Secondary and Tertiary Criteria (Table 5.1. These categories are useful as a basic framework for which additional criteria that may be defined by the user.

**Table 5.1. Criteria Categories**

| Primary Criteria | Secondary Criteria | Tertiary Criteria |
|---|---|---|
| Enumeration, $e$<br>Contextual, $c$<br>Previous Usage, $u$<br>Push-based, $p$ | Model based, $m$<br>Induction, $i$ | Time-based inference, $t$<br>Spatially-base inference, $s$ |

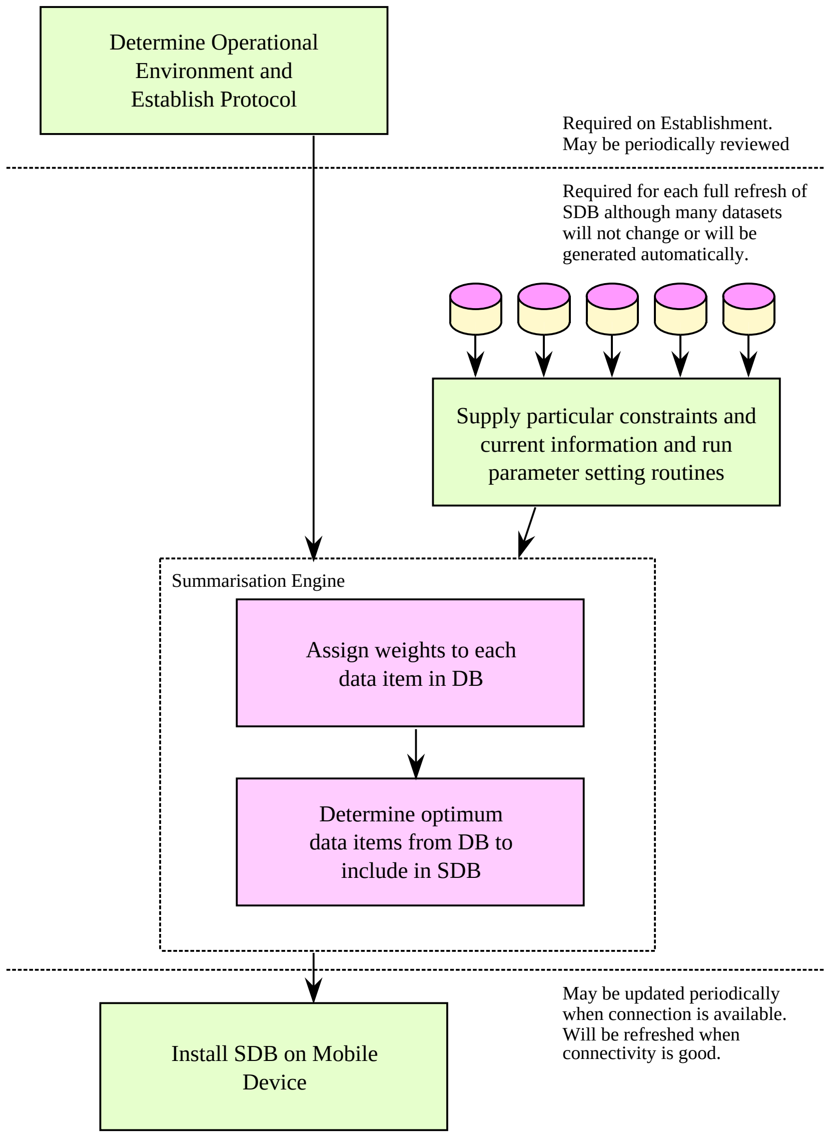

Determine Operational Environment and Establish Protocol
Required on Establishment. May be periodically reviewed
Required for each full refresh of SDB although many datasets will not change or will be generated automatically.
Supply particular constraints and current information and run parameter setting routines
Summarisation Engine
Assign weights to each data item in DB
Determine optimum data items from DB to include in SDB
May be updated periodically when connection is available. Will be refreshed when connectivity is good.
Install SDB on Mobile Device

**Figure 5.2. Conceptual View of the Summarisation Process**

The protocol and the priorities are combined through a single formula which gives a prioritisation $\mathcal{P}_d$ for each data item as shown in (Formula 5.1) and as shown in Figure 5.2. As a policy decision, any criterion not used is set to zero.

$$\mathcal{P}_d = \left[ \sum^{x} \rho_x . \phi_x \right] / ln(len_d + 1) \tag{5.1}$$

where

$x$ is one of the criteria defined in the protocol,

$\rho_x$ is the relative priority of criterion $x$ in the protocol,

**Table 5.2: Example of a criteria protocol – the setting of $\rho$ for each data inclusion criterion.**

| Criteria | | $\rho$ |
|---|---|---|
| Enumerated | $e$ | 100 |
| Contextual | $c$ | 75 |
| Model-based | $m$ | 50 |
| Inductive | $i$ | 45 |
| Previous Usage | $u$ | 65 |
| Time-based | $t$ | 20 |
| Spatially-based | $s$ | 20 |
| Push-based | $p$ | 100 |

$\phi_x$ is the assigned priority for that data item as calculated for that criterion,

$len_d$ is the size of the data item (in bits). This value is necessary to discourage the storage of overly large data items.

$\mathcal{P}_d$ provides an indication as to the importance of the data item and is thus proposed as the basis for the first stage of database summarisation.

The values of $\rho_x$ are usually not bound by any limits. They provide an indication as to the importance of individual criterion as compared to the others. Therefore as the values of $\rho_x$ become closer, the importance of the criteria will be increasingly similar. On the other hand, the value of $\phi_x$ must be within the bounds of 0 and 1, i.e., $0 \leq \phi_x \leq 1$. The following sections provide a discussion regarding the determination of the values $\rho_x$ and $\phi_x$, relative to the other criteria, and the ability of each criterion to recalculate the priority of individual attribute values. Recalculation is necessary during the runtime of a database system to ensure that database refresh is possible. This will be examined in Chapter 8.

## 5.2.1 Primary Criteria

Criteria that extracts data from the main database to the summary database through an external input are part of the category called the Primary Criteria. The following subsections discuss the types of criteria that fall under this category.

#### 5.2.1.1 Enumeration

Enumerated data are references to those items specifically indicated as useful by an agent external to the summarisation process and which indicate information that is directly recommended for representation in the summary database. It allows direct external input into the creation of the summary database and strongly encourages the summarisation process to include data which is of significance to users. Although potentially any specification can be made, the enumeration of data is commonly a horizontal specification of tuples within one or more relations. It is also possible to enumerated a list by using the 'hoarding' method(Tait, Lei, Acharya & Chang 1995). Hoarding prefetches data according to the current application being used.

> **Example:** A medical practitioner has a list of patients to be visited, and will thus enumerate a list that will be used to specify the patient data required. That is, specific tuples of a relation corresponding to the patients' details are given priorities that will encourage their inclusion in the summarisation database. Following the example given in Figure 5.3, these tuples may be from the relation corresponding to the entity PATIENT.

Since enumerated information are explicitly specified, perhaps in terms of a parameterised relational query, $\rho_e$ is likely to be assigned, for most applications, a higher value in the protocol than data specified through the other criteria. Moreover, the simplest form that $\phi_e$ is likely to take is either 0 or 1, depending on whether the data is enumerated or not. More complex methods of determination requires the user to indicate which data or group of data they feel are more important. Arbitrary numbers between 0 and 1 may then be assigned to the $\phi_e$ of each data, since the protocol is more concerned with the relative importance of data. Algorithm 1 provides an example of how to implement an enumeration criterion.

**Algorithm 1** Enumeration Criterion

```
Require: Input list ($I$) of attribute values to be enumerated and $\rho_e$
  while $I$ contains attribute values do
    Remove an attribute value ($a$) from $I$
    Assign $\phi_e(a) = 1$ to attribute value
    Assign priority, $w_e(a) = \rho_e.\phi_e$ to attribute value
    Add $a$ and $w_e(a)$ to enumerated list $E$
  end while
```

The recalculation of the total enumeration priority for an individual attribute value is simply recalculating $\rho_e * \phi_e$. $\phi_e$ will continue to take the form of 0 or 1 depending on whether the new attribute value is one that has been specified earlier by the criterion.

**Table 5.3. Example of a set of contextual rules.**

| Rules | Corresponding Relation | Corresponding Attribute | $\phi_c$ |
|---|---|---|---|
| Specialisation | PHYSICIAN | SID | 100 |
| Project | PROJECT | PID | 25 |
| Practice Location | PHYSICIAN | LOC | 50 |

#### 5.2.1.2 Contextual

Knowledge of the context of use is useful in inferring the data that may be needed by users. Thus, *contextual* criterion can be used to include data that may be useful because they are related to the user's details or environment. As such this criterion differs from the enumerated criterion which is data-centric rather than user-centric. The contextual priorities are deduced through rules that relate to aspects of a user, such as the user's medical specialisation, through to data that may be useful in that user's operation of the database.

> **Example:** A medical specialist may, by virtue of their specialisation, have a greater interest in particular aspects of a patient's case history. In addition, records of patients who have been seen recently by the medical practitioner and who are still under active treatment by others, might also be included into the summary database to ensure that the information is available if consultation is required.

In order to determine contextual priority, $\phi_c$, there must be a set of rules concerning the user (specialisation, practice, projects, etc.). This set of rules will provide a profile of the user. The assignment of $\phi_c$ due to the different contextual criteria will vary as a result of the importance placed on the data. For instance, data relating to the specialisation of the practitioner can have higher importance than the inclusion of records relating to recently seen patients. An example of this is shown in Table 5.3. An implementation example using these rules is given in Algorithm 2.

For the recalculation of the total contextual priority of an attribute value, the same set of values for $\phi_c$ and $\rho_c$ used during the initial calculation, may be used. For example, in the case of an eye specialist, assume that a patient's diagnosis data now included a cataract treatment. That is, an attribute value now has the value 'Cataract Treatment'. Then the attribute value will be assigned a value of $\phi_c$ depending on what has been assigned to the value 'Cataract Treatment'.

**Algorithm 2** Contextual Criterion

```
Require: Rules definition set $D$, Rules set $R$ regarding users and $\rho_c$
  while $R$ contains rule $r$ do
    Remove $r$ from $R$
    Extract $\phi_c(r)$
    Use $D$ and $r$ to determine search conditions for queries
    Search for attribute values ($A$) in main database using queries
    Assign priority, $w_c(A) = \rho_c.\phi_c(r)$ to attribute values
    Add $A$ and $w_c(A)$ to contextual list $C$
  end while
```

#### 5.2.1.3 Previous Usage

This criterion allows the user's previous activity to act as a guide to future activity. This is accomplished through an inspection of previous queries invoked and assigns a priority based on the frequency of access, of either the data item explicitly or the likely access through a type of heuristic. For example, association rule mining might be used to associate the use of one attribute with another or between the characteristics of one query and the next.

> **Example:** A particular user might have a preference for referring to similar cases before deciding treatment. Thus, for that user, the presence of a patient record with a particular set of conditions may also cause the inclusion of other similar cases.

The manner in which the values for $\phi_u$ can be calculated is wide. A simple technique examines the data, in terms of attribute values, accessed by previous queries issued by a particular user. By determining the frequency of each attribute value accessed, a percentage of the total number of accesses made to all attribute values may be assigned to the $\phi_u$ of each attribute value. For example, assumimg that the lifetime of a database system is divided into sessions, and each session represents an hour of the system's time, and that the frequency of access is sampled according to the transactions that has ended within a session. Then, by dividing the number of accesses to a particular attribute value by the maximum number of access made to any attribute value, it is possible to assign $\phi_u$ with a value between 0 and 1. $\phi_u = 1$ will then indicate that the attribute value has been accessed most often.

$$\phi_u = \frac{n}{n_{tot}} \tag{5.2}$$

where

$n$ is the total number of accesses made to an attribute value per session.

$n_{tot}$ is that total number of accesses made to any attribute values by a transaction within the session.

A second solution to calculating $\phi_u$ is to count the number of times an attribute is accessed instead of a particular attribute value. This will assign attribute values with a particular attribute to have the same $\phi_u$ value. $\phi_u$ can again be calculated by Equation 5.2 except that $n$ and $n_{tot}$ now refer to attributes instead of attribute values. The disadvantage to using this solution is that priority is assigned to attributes instead of attribute values and redundant values may be selected. However, such a method may be useful when transactions specify different keys for the same attributes. For example, querying for the name of different items. Algorithm 3 provides an example to implement this criterion.

**Algorithm 3** Previous Usage Criterion

```
Require: Set of previously accessed attribute values $PU$ and $\rho_u$
  while $PU$ contains attribute value $pu$ do
    Remove $pu$ from $PU$
    Determine attribute $A(pu)$ of $pu$
    Counter $C(A(pu)) = C(A(pu)) + 1$
    Add $A(pu)$ to previous usage list $U$
  end while
  Determine most accessed attribute, $max$
  for all $A(pu)$ in $U$ do
    $\phi_u(pu) = C(A(pu))/max$
    Assign priority, $w_u(pu) = \rho_u.\phi_u(pu)$ to attribute value
    Add $w_u(pu)$ to $U$
  end for
```

The recalculation of previous usage criterion on an attribute value occurs at the end of every session regardless of whether an update has occurred or not. This recalculation uses the same formula for $\phi_u$ and $\rho_u$ as mentioned above (and in this case, Equation 5.2). In the case where an update has been made to an attribute value, the value $n$ for that attribute value will reset to one. This will indicate that the update process has accessed the new attribute value.

#### 5.2.1.4 Push-based Approaches

The push-based criterion indicates that the data has been determined important by the server and should be communicated to the clients. In this respect it can be used as a mechanism for partially updating a summary database without full synchronisation. Such data may include updates that the server has received (such as recent pathology laboratory results). It can also be used to manually override decisions made by the summarisation engine. Algorithm 4 provides an implementation of the push-based criterion.

**Algorithm 4** Push-based Criterion

```
Require: Set of push content $C$, associated $\phi_p(C)$ and $\rho_p$
  for all $c$ in $C$ do
    Determine attribute values $A$ for $c$
    Assign priority, $w_p(A) = \rho_p.\phi_p(A)$ to attribute values
    Add all $A$ and $w_p(A)$ to push-based list $P$
  end for
```

### 5.2.2 Secondary Criteria

Criteria using data that has been extracted by the Primary Criteria in order to determine any new data for the summary database are part of the category called Secondary Criteria. The following subsections discuss the criteria that fall under this category.

#### 5.2.2.1 Model or Schema-based

The model or schema-based criterion is based on the relationships implied through the data model. That is, data flagged for inclusion by this criterion is related to other data through the structure described in the database schema. The inclusion of data into the summary database using this criteria depends on the data extracted by any Primary Criteria.

The model-based priority, $\rho_m$, determines its importance relative to the other criteria (if no model is available then this is set to zero). It is important to note that as the modelled relationship to the data generated by the Primary Criteria becomes more distant, the importance of the data is assumed to decrease. Thus, the weight of $\phi_m$ would decrease as the modelled distance increases. An example formula for $\phi_m$ may be given by:

$$\phi_m = k^{-(a-1)} \tag{5.3}$$

where $k \geq 1$ is a constant and $a > 0$ is the length of the shortest path from the data under consideration to the Primary Criteria data that it is associated to.

It is also possible that an attribute value may have relationships with multiple Primary Criteria data. In this case, $\phi_m$ is calculated for each relationship and averaged together to form the final $\phi_m$ Additionally, $\phi_m$ is always zero when none of the Primary Criteria are present. That is, $\phi_m = 0$ when $\phi_e$, $\phi_c$, $\phi_u$ and $\phi_p$ are all equal to zero.

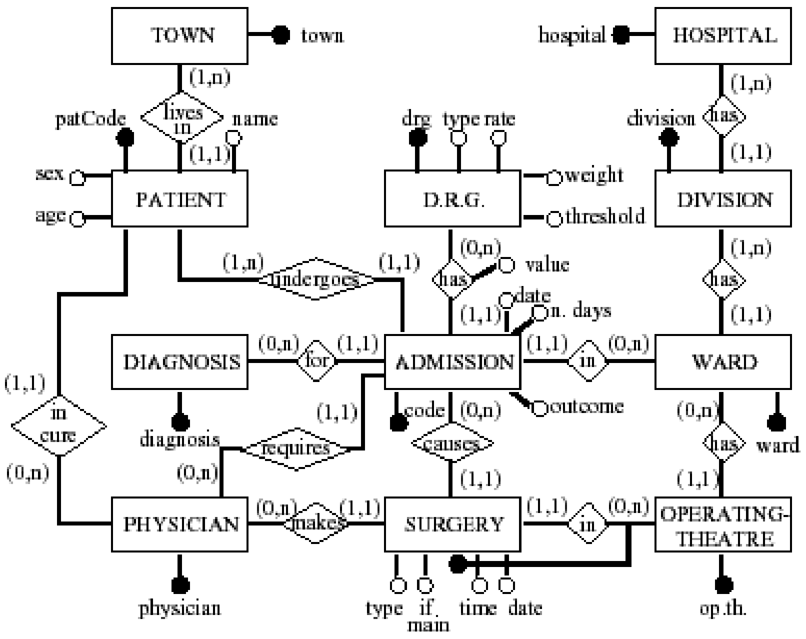


**Figure 5.3. A medical E-R scheme, after (Golfarelli et al. 1998)**

**Example:** Following the example in Section 5.2.1.1, and the associated Entity-Relationship diagram for the example in Figure 5.3, it can be seen that two important entities exist- PATIENT and ADMISSION that provide information regarding patient details and details of their hospital admissions, respectively. By observation, tuples from ADMISSION are related to PATIENT through the relationship undergoes. Assuming these are annotated as useful through primary criteria and are given weights that indicate certain usefulness.

Using the model-based criterion, the data residing in entities WARD, DIVISION and HOSPITAL, will be given incrementally lower weightings depending on the number of relations they are from the entity PATIENT.

An implementation of model-based criterion is given in Algorithm 5. The recalculation of Secondary Criteria is more complex than those techniques presented for Primary Criteria. This is because priority values of Secondary Criteria,

**Algorithm 5** Model-based Criterion

```
Require: enumeration list $E$, contextual list $C$, push-based list $P$, number of
  relations $n$ and $\rho_m$
  Combine lists ($E$,$C$,$P$) together into primary criteria list $PR$
  Sort list into its originating relations $R$
  for $x = 1$ to $n$ do
    Determine proximity of other relations to $R(x)$
    Determine maximum proximity to $R(x)$ ($max_r$)
    for $y = 1$ to $max_r$ do
      for all relations $y$ away from R(x) do
        Determine attribute values $A$ related to all $r$ in $R(x)$
        $\phi_m(A) = 2^{-y}$
        Assign priority, $w_m(A) = \rho_m.\phi_m(A)$ to $A$
        if Model-based list $M$ contains attribute values in $A$ then
          if $w_m(a) > w_m(m)$ for all $a$ in $A$ and $m$ in $M$, and $a = m$ then
            Replace $w_m(m)$ with $w_m(a)$
          end if
        else
          Add all $A$ to $M$
        end if
      end for
    end for
  end for
```

for an attribute value are influenced by data that has already been assigned a Primary Criteria value. So after an update, changes can occur to the model-based priority value, of both the updated attribute value and any attribute values it is related to, through the database schema. For updated attribute, the model-based priority value is changed if the value is no longer related to any Primary Criteria, or is now related to other attribute values. The Equation 5.3 is used to recalculate the model-based priority for that attribute value. Additionally, the new updated value may now be related to other attribute values. In this case, a search for other related attribute values would then be required if the updated value had been assigned a Primary Criteria value. On the other hand, if the updated value is no longer Primary Criteria data, a search is required to determine the attribute values that were related to the attribute value prior to an update. In a related work, Badrinath et al. (1999) examined referential integrity to cluster data and create fragments for caching.

#### 5.2.2.2 Induction

This criterion enables the use of inductive rules to specify the inclusion of data. By inspection of the data contained within the main database, it is possible to derive rules to indicate data that are associated with each other. Data mining techniques, such as Association Rule Mining (ARM) as typified in the Apriori algorithm (Agrawal, Imielinski & Swami 1993) and its many successors, are useful for the derivation of inductive rules. ARM involves the identification of relationships between items that occur frequently together and is then be used to imply the storage of other items within the database.

> **Example:** Within the medical records of a patient, assuming there is an association between a disease and the pathology tests used to diagnose and monitor this disease. Depending on the strength of this association, this would imply that the appropriate pathology results of the patient suspected with this disease would then be flagged for storage in the summary database.

Similar to model-based criterion, data are related to any Primary Criteria or Secondary Criteria already generated. They are thus dependent on those data, where the condition ($\phi_i = 0$), again, applies when no Primary Criteria are present. Note that the importance of each rule used can be specified, which could be used to determine the correct values for $\phi_i$. Algorithm 6 shows an implementation for the induction criterion.

Recalculation of induction-based criterion for an attribute value is similar to model-based criterion. That is, it requires a search to determine if the attribute value is still affected by other attribute values, and if the new attribute value will affect other attribute values.

### 5.2.3 Tertiary Criteria

Finally, tertiary criteria is a category of criteria that involves specific data types within the summary database. They are used to place more emphasis upon certain attribute values, depending on how the priority weights are calculated for the criterion.

**Algorithm 6** Induction Criterion

```
Require: enumeration list $E$, contextual list $C$, push-based list $P$, model-based
  list $M$, set of inductive rules $R$, set of $\phi_i(R)$ associated to $R$ and $\rho_i$
  Combine lists ($E$,$C$,$P$) together into primary criteria list $PR$
  Combine lists ($PR$,$M$) together into a list $L$
  while $L$ contains attribute values $l$ do
    Remove $l$ from $L$
    if $pr$ exists in $R$ then
      Determine all attribute values $A$ from rules $R$
      Assign priority $w_i(A) = \rho_i.\phi_i(R)$ to $A$
      for all Attribute value $a \in A$ do
        if Induction list $I$ contains $a$ AND $w_i(a) > w_i(i)$ where attribute value
        $i = a$ then
          Replace $w_i(i)$ with $w_i(a)$
        else
          Add attribute value $a$ and $w_i(a)$ to $I$
        end if
      end for
    end if
  end while
```

#### 5.2.3.1 Time-based Inference

This criterion makes the assumption that recent events are of more importance. In the modelling of time-based data there are a number of characteristics to be considered (Roddick & Patrick 1992, Snodgrass 1987). In particular, two temporal dimensions are identified as important when an event occurs, namely *transaction* and *valid* time. The former refers to the time that the event is recorded into the database, allowing the user to *rollback* the database to an earlier view. In the context of this thesis, transaction time is less relevant since the creation of a summary database is based on the data that are stored being useful and accurate to the user. The latter, *valid time*, refers to the time that events have occurred in reality, and facilitates the post or predating of changes to the database. The *valid time* of an event is likely to be of interest. Thus, time-based criterion focuses on an event's valid time reference.

> **Example:** A patient had an episode relating to a fractured arm some time ago. More recently, the patient made an appointment for a consultation with the medical practitioner for a cough. Apart from the relative association between a fracture and a cough (dealt with through the inductive priority (i.e., the value of $\rho_i$)), the time since the first event will mean that it is less likely to be of relevance and would be given a lower prior-

ity. On the other hand, a sore back that the patient mentioned during a more recent consultation would be given a higher priority (and thus may be included in the summary database).

$\phi_t$ indicates the relative importance of the data item due to its temporal information. $\rho_t$ is a priority specified as part of the protocol and indicates the importance of the time-based criterion. Since time is continuous, $\phi_t$ may be calculated by either one of the following functions:

- **Stepwise**. This assumes that data has the same level of interest (and priority) until changed.
- **Linear or Continuous Change Functions**. These functions assume some degradation of interest over time.

In both calculations, it is assumed that only attribute values with at least one associated timestamp attribute will have time-based inference criterion. It is therefore likely that attribute values within a tuple will have the same time-based priority value. So only attribute values that have at least a Primary Criterion will be recalculated. This involves recalculating $\phi_t$ for the new attribute value according to whichever function was initially specified. An implementation of time-based criterion is shown in Algorithm 7.

**Algorithm 7** Time-based Criterion

```
Require: enumeration list $E$, contextual list $C$, push-based list $P$, previous usage list $U$, model-based list $M$, induction list $I$s and $\rho_t$
  Combine lists ($E$,$C$,$P$,$U$,$M$,$I$) together into primary and secondary criteria list $PS$
  Determine current date and time, $time$
  for all Attribute value $ps \in PS$ do
    Determine associated timestamp $d$ for $ps$
    if $x = (time - d) > 0$ then
      $\phi_t(ps) = 2^{(-x)}$
      Assign priority $w_t(ps) = \rho_t.\phi_t(ps)$ to $ps$
      Add $ps$ and $w_t(ps)$ to time-based criterion list $T$
    end if
  end for
```

#### 5.2.3.2 Spatially-based Inference

In the same way that time-based criterion assumes recent events are more important than those in the past, spatially-based inference assumes that physically or geographical closer events are of more interest than those more spatially distant. Thus, by using location knowledge, it is possible to give a higher priority to data that corresponds to that location, as there is an assumed higher probability that data relating to that area will be accessed.

> **Example:** A travelling medical practitioner is visiting patients in a remote location. In addition to the details relating to the (enumerated) patients, information relating to other patients residing in the same area could also be stored.

As with time-based criterion, where attribute values require at least one associated timestamp attribute, spatially-based criterion is calculated for those attribute values with associated spatial attributes, such as an address. $\phi_s$ is then be calculated using the formula:

$$\phi_s = c^{-d} \tag{5.4}$$

Where $c \geq 1$ is a constant, and $d$ is the shortest distance (geographically) that the attribute value is from the location of the target data. This indicates that the closer the attribute values are to the target's location, the more important the attribute value will be. For recalculation, $\phi_s$ is only required when an attribute value has Primary Criteria priority. This recalculation would then determine a new value for $\phi_s$ . Algorithm 8 shows an implementation of the spatially-based criterion.

**Algorithm 8** Spatial-based Criterion

```
Require: enumeration list $E$, contextual list $C$, push-based list $P$, previous usage list $U$, model-based list $M$, induction list $I$s and $\rho_s$
  Combine lists ($E$,$C$,$P$,$U$,$M$,$I$) together into primary and secondary criteria list, $PS$
  Determine a set of origin points, $O$
  for all Attribute value $ps$ in $PS$ do
    if $ps$ has associating spatial data, $d$ then
      Shortest distance $s = min(d - O)$
      $\phi_s(ps) = 2^{(-s)}$
      Assign priority $w_s(ps) = \rho_s.\phi_s(ps)$ to $ps$
      Add $ps$ and $w_s(ps)$ to time-based criterion list $S$
    end if
  end for
```

### 5.2.4 User-defined

The criteria described are not an exhaustive list of those that could be used in the COSMOS process. COSMOS also allows additional user-defined criteria to be included. The main condition that any user-defined criteria must adhere to is that it must fall into the behaviour of any one of the three categories (Primary, Secondary or Tertiary).

## 5.3 Summarisation Stage Two

Having determined the priorities of each data item, the task is to construct a summary database such that the organisation of the summary is not overly complex. It must be possible to calculate rapidly whether a query to the summarised database is able to return the same response as the same query to the original database. That is, whether, for a given query $Q$

$$Q(DB) \equiv Q(SDB) \tag{5.5}$$

where $SDB = \Psi(DB)$ and $\Psi$ is the summarisation function.

If $\Psi$ is able to be stored, for example, as (the equivalent of) a relational algebraic expression (i.e., a view), then query rewriting techniques (with extensions) could be used to evaluate the equivalence. For example, given the priorities shown in Figure 5.4, the optimum summarisation, as shown by the colour, might be

$$\sigma_{Id \in \{10129,10187\}}(RelA) \oplus \pi_{\{ID,AttA\}}(RelA) \tag{5.6}$$

where $\oplus$ is a combination function.

With traditional databases, the objective is to provide query answers that are logically complete and sound. However, summary databases are by their nature associated with a loss in data and are thus incomplete but sound. That is, these summary databases may provide some of the solutions available to a complete main database. Consequently, it is not always possible to provide both sound and complete answers. Hence, in the second stage of COSMOS, views may be constructed to include attribute values that have an associated priority value exceeding a certain threshold. The threshold is usually determined such that the size of the attribute values summed together is within a specified size. The summary database then has views such as that shown in Figure 5.5, where

| RelA | Id | AttA | AttB | AttC |
|---|---|---|---|---|
| | 10002 $^{1.2}$ | D34 $^{1.7}$ | 23,000 $^{.7}$ | 76, The Avenues ... $^{.6}$ |
| | 10077 $^{1.2}$ | D32 $^{1.7}$ | 24,500 $^{.7}$ | 1, The Arches ... $^{.6}$ |
| | 10093 $^{1.2}$ | D34 $^{1.7}$ | 29,000 $^{.7}$ | 19, Boulevard Tce ... $^{.6}$ |
| | 10129 $^{1.8}$ | D32 $^{2.1}$ | 23,500 $^{1.4}$ | c/o PO Box 15, ... $^{1.1}$ |
| | 10165 $^{1.2}$ | D33 $^{1.7}$ | 28,000 $^{.7}$ | 1232, Great South Rd ... $^{.6}$ |
| | 10184 $^{1.2}$ | D33 $^{1.7}$ | 26,250 $^{.7}$ | 992, Great South Rd ... $^{.6}$ |
| | 10187 $^{1.8}$ | D32 $^{2.1}$ | 26,250 $^{1.4}$ | 33, Maple Street ... $^{1.1}$ |
| | 10211 $^{1.2}$ | D39 $^{1.7}$ | 23,000 $^{.7}$ | 244, The Avenues ... $^{.6}$ |

**Figure 5.4. Example relation with summarisation priorities**

attribute values that are not included are substituted by a local null. The concept of local nulls are discussed in Chapter 6.

| RelA | Id | AttA | AttB | AttC |
|---|---|---|---|---|
| | 10002 | D34 | Lnull | Lnull |
| | 10077 | D32 | Lnull | Lnull |
| | 10093 | D34 | Lnull | Lnull |
| | 10129 | D32 | 23,500 | c/o PO Box 15,... |
| | 10165 | D33 | Lnull | Lnull |
| | 10184 | D33 | Lnull | Lnull |
| | 10187 | D32 | 26,250 | 33, Maple Street ... |
| | 10211 | D39 | Lnull | Lnull |

**Figure 5.5. View of relation in Figure 5.4 with local nulls**

### 5.3.1 Storage Maps

As well as constructing the views during this second stage 2, it is also necessary to provide a descriptive representation of the structure of the resultant summary database to allow the query processor to determine if a given query can be answered. Many methods capable of representing a view. The adopted format is through the use of Storage Map (SM).

For the first stage of the summarisation process, each data item is assigned a particular priority that is used to determine the items selected for inclusion. In addition, as the summary database is built, a second parallel set of matrices are developed that hold a boolean flag corresponding to each of the data items within the database. This second set of matrices, together with the schema itself, are known as the SM. It is kept after summarisation to enable query answering and transaction processing. A SM that corresponds to the relation in Figure 5.5, is shown in Figure 5.6

| RelA | Id | AttA | AttB | AttC |
|---|---|---|---|---|
| | 10002 | 1 | 0 | 0 |
| | 10077 | 1 | 0 | 0 |
| | 10093 | 1 | 0 | 0 |
| | 10129 | 1 | 1 | 1 |
| | 10165 | 1 | 0 | 0 |
| | 10184 | 1 | 0 | 0 |
| | 10187 | 1 | 1 | 1 |
| | 10211 | 1 | 0 | 0 |

**Figure 5.6. An example of a storage map**

The main advantages of using SM to store descriptions are:

- a fixed, relatively short, description length. Since a bit is used to represent each value within the database, it is easy to determine the length of the description,
- for many cases (although not always) a lower storage requirement than the equivalent algebraic expression, and
- a fast way to determine if queries could be answered. Simple logical binary operators can be used to quickly determine if a query may be answered.

The main disadvantage of using a storage map is that the technique may not provide the most optimum description of the summary database. That is, since SMs are of fixed length, it is not possible to reduce the size of the description without using compression techniques. If all keys are stored within the summary, a larger storage requirement is required.

There are two cases that must be considered when creating a storage map. The first is to store all the keys found in the main database. For large databases, this case is expensive since they are likely to contain many keys. Additionally, new keys added to the main database must also be reflected in the summary database and thus frequent synchronisation of keys will be required. Since all keys in the summary database reflect the main database, it is possible to provide a negative answer when queries refer to keys not present in the summary database.

The second approach is to store only selected keys into the summary database. One method of this is to determine the primary keys of the tuples that are stored in the summary database. This would then identify which keys are necessary in the storage map. This approach resolves the size problem and does not require frequent synchronisation. However, reference to key values not held requires querying the centralised database to ensure that all data will be retrieved.

## 5.4 Discussion

A mobile database is as useful as its contents. However, storage constraints on mobile devices limit the amount of information that may be stored. Thus, context sensitive techniques are required to effectively determine the stored information. COSMOS introduces a protocol that uses context sensitive criteria to rate each attribute value according to their importance. This rating is subsequently used to store the required information into the mobile database and create a SM. The SM may then be used to assist in queries and transactions. The construction and uses of the SM is discussed in Section 8.1.

# Chapter 6

# Local Nulls

Codd and Zaniolo gave the semantics of null values as three-fold – *value unknown*, *value inapplicable* and *no information* (Codd 1970, Zaniolo 1982) while the ANSI/SPARC Interim Report (ANSI/X3/SPARC 1975, p. IV-28) gave fourteen different reasons as to why a null value might appear. A variety of research has since expanded on this – see particularly (Biskup 1983, Codd 1986, Codd 1987, Date 1986, Imieliński & Lipski 1984, Reiter 1986, Roth et al. 1989, Vassiliou 1979). In this thesis, an additional definition for the null value in mobile and distributed databases is required (Chan & Roddick 2003) – that of a 'local null' – which is not so far investigated and not covered by existing interpretations.

Within a summarised database, there may be attributes for which a value exists only in the global database. Moreover, the lack of connectivity in a mobile environment may result in this summarised database being the best that is available. The context of our work is thus to maximise the usability of the data available – i.e., the maintenance of *maximal completeness* when summarised databases are used in a low capacity mobile environment. We use the term *summarised* here to refer to any database which holds, in whatever form may be appropriate, a fragment of some 'relatively global' database. The term 'relatively global' allows for a hierarchy of fragments.

Local nulls can be loosely defined as items that are not available locally, but *may* be available from the global database. During periods of good communication nulls can be handled by passing a request to the global database. However, in cases where the global database is not accessible[2] it would be misleading to return

[1]A version of this chapter appears as part of Chan and Roddick (2006)

[2]This is not an uncommon occurrence with mobile devices. Network inaccessibility may be caused by disconnection, low priority in partial or weak connection mode or intentional

a 'global' null value. Null values that are (authoritatively) held in some relatively global database as 'global nulls' and while 'local' null values are non-authoritative values held on distributed and/or mobile devices.

This chapter discusses a novel extension to the current notion of null values to include the semantics of nulls found in mobile and distributed systems. It introduces local nulls in terms of amendments to the relational algebra and examines its impact on the query languages such as SQL. Interestingly, despite the orthogonality of relations in query languages such as SQL, the impact of a local null can be different depending on where in the statement the local null appears.

This chapter is structured as follows. Section 6.1 discusses the research to date and provides more details on our motivation for this problem. Section 6.2 then introduces the notation that will be used within the paper. Section 6.3 examines the relational algebra in terms of local nulls. Section 6.4 discusses the changes to query languages such as SQL to include local nulls. Finally, section 6.5 will provide a conclusion and a discussion of the use of local nulls.

## 6.1 Previous Research and Motivation

### 6.1.1 Literature Survey

Much of the research on the semantics of null values in relational databases dates back to the 1970s and 1980s (Codd 1970, Lacroix & Pirotte 1976, Maier 1983, Zaniolo 1984, Roth et al. 1989, Imieliński & Lipski 1984, Vassiliou 1979). The two definitions of nulls as given by Codd are missing and applicable, and missing and inapplicable (Codd 1970) and Zaniolo (1992, 1984) later proposed a third definition as, essentially, a lack of knowledge about the attribute's applicability, or *no information*.

To handle null values, various logical approaches have been developed. For example, the commonly-used three value logic includes true, false (often by virtue of a value's absence - q.v. the closed-world assumption (Reiter 1978)), and a maybe value which indicates that the results may be true (Yue 1991, Codd 1979). A four value logic has also been proposed which includes an additional truth value, which represents the outcome of evaluating expressions which have inapplicable values (Codd 1986, Gessert 1990).

disconnection through power management because of limited battery capacity (Chan & Roddick 2005).

Approaches to accommodating null values in practical systems include the work of Motro (1988) who uses the ideas of conceptual closeness *fill the vacancies* represented by a null value and Roth *et al.* who aim to accommodate nulls in NF$^2$ databases (Roth et al. 1989). Null values have also been studied in relation to schema evolution and integration (Kim & Seo 1991, Roddick 1995).

However, there has been little research being undertaken in terms of missing data / nulls in distributed and mobile databases. Much of the reason for this is that there is little common agreement as to how to deal with such values. However, some of the techniques introduced to deal with null values in relational database are also applicable to distributed databases. That is, in many cases the evaluation of any query on a distributed database system as a whole may be viewed as an evaluation of that query on a single relational database with all the data of the distributed one. Thus, in some cases, although not all, specialised techniques to handle nulls in distributed databases can be avoided. Moreover, there is research that introduces techniques to allow for the approximation of incomplete data using fuzzy rules (Chen & Chen 2000).

Finally, it should be noted that many of the techniques for handling data (including nulls) in distributed databases are often applicable to mobile databases, especially they are viewed as extensions to distributed databases. In such cases, many designers are able avoid developing new techniques to dealing with problems in mobile databases specifically. Similarly, modifications to techniques developed for mobile systems are often applicable to distributed systems.

### 6.1.2 Motivation

As motivation for using local nulls, an ongoing example of a medical practitioner visiting patients is adopted. In this example, the practitioner carries a PDA which stores a summarised version of the main database. It is assumed that the main database contains all information available to the practitioner at the medical clinic, and the summarised database stores an optimum subset of (what has been estimated to be) relevant information[3].

Should the main database lack information such as patient's details these are then stored into the main database as null values. Thus, within the summary database, the same information, if selected for inclusion, would also be represented

[3]Note that 'relevant information' can be determined using the context-sensitive framework, COSMOS, as discussed in an earlier paper (Chan & Roddick 2003) or by some other means.

as null values. Since the summary database is created using relevancy, much irrelevant data is excluded and, in the absence of local nulls, these missing data would also be represented by null values. In such cases, while the summarised database is disconnected from the clinic's network, it will not be possible to differentiate between data that are available somewhere in the network and data that are globally unavailable. Since the existence, or otherwise, of the data is known when the summary database is populated, by introducing the concept of a 'local null', a disconnected summary database would be able to distinguish between them.

There are a number of benefits to this. For example, in a mobile, wireless or large area distributed environment, where the cost of communication are releatively high, by knowing which data is available it is possible to request only specific data, and thus reducing the traffic over the low bandwidth communication network. That is, by examining the query, such as one that displays the patient's details, it is possible to create sub-queries that resolve the local nulls. These sub-queries may then be forwarded to the main database, thus allowing the practitioner to vary the accuracy of the results. That is, lower accuracy would result in faster response time. On the other hand, if higher accuracy is required, a slower response time (and fewer local nulls) would be returned as the querying is done mostly at the main database.

## 6.2 Notation

Maier (1983) examined the presence of unknown values in relational databases, which have been adopted here as a basis for our modifications to include the notion of local null values. This section presents the amended notations and provides a comparison to Maier's work.

For the purposes of this paper, local and global null values will be represented as $\varphi$ and $\omega$, respectively.

A tuple $t$ in a local relation is a *locally complete tuple* (LC-tuple) if it holds all data that also exists in the equivalent attributes within the global database[4]. Conversely, a *locally partial* tuple (LP-tuple) contains one or more local nulls. LC-tuples will not contain any local nulls but can, of course, contain global nulls. LP-tuples contains global $\omega$ and local $\varphi$ nulls.

[4]Note that the local and global schemata may differ. For example, the local database may possess a restricted subset of attributes.

The designation of LC and LP-tuples can be extended to include Maier's notation (1983), in which a tuple may be designated as a *partial* tuple or a *total* tuple depending on whether global nulls exists in the tuple. When an attribute value, $t(A)$, is not a local null it is considered to be equal to its representation in the global database, $t(A)^L \downarrow$, where $t(A) \downarrow$ defines an attribute value that is not a global null. Thus, for a set of attributes $X$, $t(X)^L \downarrow$ implies $t(A)^L \downarrow$ for every attribute $A \in X$. A simplified notation, $t^L \downarrow$, is then used to define $t$ as a locally complete tuple, while a complete tuple consists of no nulls of either type. Similarly, for tuples, $t$ and $u$ defined over the same schema, $t$ *locally subsumes* $u$, $t \geq^L u$ if $\forall\ u(A)^L \downarrow$, $u(A) = t(A)$. If $t \geq^L u$ and $t^L \downarrow$, then $t$ is a *local extension* of $u$, $t \downarrow\geq^L u$. For example, the tuple $< a, \varphi, \varphi >$ is locally subsumed by $< a, b, \omega >$, and in this case, the tuple is also locally extended by $< a, b, \omega >$. In comparison to Maier's work, tuple $< a, b, \omega >$ is subsumed and extended by $< a, b, c >$.

A relation $r$ is a *locally complete relation* (an LC-relation), $r \downarrow^L$, when all its tuples are locally complete tuples, and a *locally partial* relation (an LP-relation) when its tuples contains one or more local nulls. For a relation scheme R, $Rel \uparrow (R)^L$ is a set of all locally partial relations over $R$, while $Rel(R)^L$ is the set of all locally complete relations over $R$. For relations $r$ and $s$ over $R$, $r$ *locally subsumes* $s$, denoted $r \geq^L s$, if $\forall t_s \in s, \exists t_r \in r : t_r \geq^L t_s$. If $r$ is a locally complete relation, then $r$ is a *local extension* of $s$ if every tuple of $s$ is subsumed by at least one tuple of $r$, denoted $r \downarrow\succeq^L s$, and it is a *local completion* of $s$ if every tuple of $s$ is subsumed by exactly one tuple of $r$, denoted $r \downarrow\geq^L s$. For example in Table 6.1, $r$ is a local extension of $s$ since both tuple $< d, e, f >$ and $< d, e, \omega >$ subsume $< d, e, \varphi >$, while $p$ is a local completion of $s$.

**Table 6.1: Example of Local Extension and Local Completion. $r$ is a local extension of $s$, while $p$ is a local completion of $s$.**

| s | (A | B | C) |
|---|---|---|---|
| | a | b | c |
| | d | e | $\varphi$ |
| | k | l | m |
| | g | $\varphi$ | i |

| r | (A | B | C) |
|---|---|---|---|
| | a | b | c |
| | d | e | $\omega$ |
| | d | e | f |
| | k | l | m |
| | g | h | i |

| p | (A | B | C) |
|---|---|---|---|
| | a | b | c |
| | d | e | $\omega$ |
| | k | l | m |
| | g | h | i |

Constraints on global nulls do not usually appear in any component of a candidate key (Maier 1983). Similarly, any component of a candidate key for any summary databases should not contain any local nulls. This is important as primary keys are used not only as object identifiers but as indexes to retrieve information when data is lacking from the summary database.

## 6.3 Local Nulls and Relational Algebra

Relational algebra provides a set of operations to be used in the manipulation and retrieval of data from a database (Codd 1970). As with global nulls, the presence of local nulls in relations requires an extension to current relational algebra.

### 6.3.1 Set Theory Operations

Two global relations, $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$, are considered union compatible, and set operations can be applied, if they have the same degree of $n$ and if $\forall i \in n : dom(A_i) = dom(B_i)$. For LP-relations where local nulls exist, this thesis shows that the use of set operators are plausible and that the results gracefully degrade as local nulls are resolved. Since local nulls are extensions of the traditional null concept, it is reasonable to conclude that set theory operations are possible for LP-relations.

**Table 6.2. Union Compatible Tables**

| r | (A | B | C) |
|---|---|---|---|
| | a | b | c |
| | d | j | $\varphi$ |
| | $\varphi$ | l | m |
| | g | $\varphi$ | i |

| s | (A | B | C) |
|---|---|---|---|
| | a | $\varphi$ | $\varphi$ |
| | $\varphi$ | j | $\varphi$ |
| | g | h | $\varphi$ |

**Table 6.3. Operations using null substitution principle**

| $r \cup s$ | (A | B | C) |
|---|---|---|---|
| | a | b | c |
| | d | j | $\varphi$ |
| | $\varphi$ | l | m |
| | g | h | i |

| $r \cap s$ | (A | B | C) |
|---|---|---|---|
| | a | b | c |
| | d | j | $\varphi$ |
| | g | h | i |

| $r - s$ | (A | B | C) |
|---|---|---|---|
| | $\varphi$ | l | m |

The conventional set theory operations of union $\cup$, intersection $\cap$ and set difference $-$ operate over two union compatible tables $r$ and $s$ (see Table 6.2):

$$\text{union:} \quad r \cup s = \{t | t \in r \vee t \in s\} \tag{6.1}$$

$$\text{intersection:} \quad r \cap s = \{t | t \in r \wedge t \in s\} \tag{6.2}$$

$$\text{set difference:} \quad r - s = \{t | t \in r \wedge t \notin s\} \tag{6.3}$$

For these three operations, only duplicate tuples will be removed from the final relation. However, for relations with global nulls, Codd introduced the *null substitution principle* to reduce redundancy in the relation (Codd 1979). This was further discussed in (Zaniolo 1984, Biskup 1983, Maier 1983) for global nulls. For LP-relations, this principle is also applicable to remove redundancy with respect to local nulls instead of global nulls. That is, if for two tuples $t$ and $u$ such that $t \geq^L u$, then $u$ will be removed from the relation (see Table 6.3 for examples).

In related work, Rice and Roddick (2000) propose a similar technique for reducing redundancy such that only the more informative tuple is kept. In their technique, however, tuples are not required to be locally or globally partial.

In cases where the primary keys of a relation are available, it is possible to exploit the keys, introducing 'keyed' operations. Essentially, the keyed operations make it explicit when two tuples represent the same object. For example, suppose there are two observations of a white car travelling at speed. One observer states that there were two passengers while the other is unsure. Given that white cars are common, it can not be assumed there was only the one car and thus merge the two observations. However, if the registration number of the cars were recorded, such an assumption can be made and the two observations merged.

Given that $\in^L$ is an extension of $\in$ for global nulls, then for a relation $r$ and its corresponding LC-relation, $r \downarrow^L$, $t \in^L r$ iff there exists a tuple $t_1 \in r \downarrow^L$ such that $t_1 \geq^L t$. Thus, a set of $n$ tuples, $\{t_1, t_2, ..., t_n\}$, represent a unique key set as $\{t_1, t_2, ..., t_n\}^k$ where $t_i(k) \neq t_j(k) \forall i, j$ in $n$ and $k$ is the set of primary key attributes.

Thus given two tuples in different union-compatible relations, defined over the schema $R$, with the same primary keys:

$$\text{keyed union:}$$
$$r \cup_k s = \{t | t \in^L r \vee t \in^L s\}^k \tag{6.4}$$
$$\text{keyed intersection:}$$
$$r \cap_k s = \{t | t \in^L r \wedge t \in^L s\}^k \tag{6.5}$$
$$\text{keyed set difference:}$$
$$r -_k s = \{t | t \in^L r \wedge t \notin^L s\}^k \tag{6.6}$$

Conflicting tuples can result from such operations, if unique keys are not identified in union compatible relations. That is, tuples conflict when, for a primary attribute $A_p$, the primary keys for two tuples are the same, while other

**Table 6.4. Value Evaluation of $x \cap_k y$ and $x \cup_k y$**

| | | *x* Value 'A' | Value 'B' | $\varphi$ | $\omega$ |
|---|---|---|---|---|---|
| | Value 'A' | A | Conflict | A | A |
| y | Value 'B' | Conflict | B | B | B |
| | $\varphi$ | A | B | $\varphi$ | $\omega$ |
| | $\omega$ | A | B | $\omega$ | $\omega$ |

non-null values are not. Since the tuple conflicts, the relations would then be considered non-union compatible and the keyed operation would fail. This union compatibility is different in that it relates to data-centric compatibility instead of structural compatibility. Table 6.4 examines the evaluation of two values, $x$ and $y$, when used in keyed union or intersection given that the primary keys match. For example, if $x$ is the value 'A' and $y$ is a local null, then the evaluated value is 'A'.

'Keyed' operations are a generalised form of conventional set operation. That is, if $r$ is a LC-relation, from observation it can be seen that $t \in^L r$ implies $t \in r$, and $t \notin^L r$ implies $t \notin r$. Similarly, if $r$ consists of set $n$ tuples, then $\{t_1, t_2, ..., t_n\}^k$ implies $\{t_1, t_2, ..., t_n\}$. Thus,

$$\begin{aligned} r \downarrow^L \cup_k s \downarrow^L &= \{t | t \in^L r \downarrow^L \vee t \in^L s \downarrow^L\}^k \\ &= \{t | t \in r \downarrow^L \vee t \in s \downarrow^L\} \\ &= \{t | t \in r \downarrow^L \vee t \in s \downarrow^L\} \end{aligned}$$

$$\begin{aligned} r \downarrow^L \cap_k s \downarrow^L &= \{t | t \in^L r \downarrow^L \wedge t \in^L s \downarrow^L\}^k \\ &= \{t | t \in^L r \downarrow^L \wedge t \in^L s \downarrow^L\} \\ &= \{t | t \in r \downarrow^L \wedge t \in s \downarrow^L\} \end{aligned}$$

$$\begin{aligned} r \downarrow^L -_k s \downarrow^L &= \{t | t \in^L r \downarrow^L \wedge t \notin^L s \downarrow^L\}^k \\ &= \{t | t \in^L r \downarrow^L \wedge t \notin^L s \downarrow^L\} \\ &= \{t | t \in r \downarrow^L \wedge t \notin s \downarrow^L\} \end{aligned}$$

### 6.3.2 Select and Project Operations

The select operation selects a number of tuples that satisfy a certain condition, as denoted by $\sigma_{<condition>}(R)$ (Elmasri & Navathe 2000). Given $R$ is a LP-relation, the same form may be used, i.e $\sigma_{<condition>}(R)$.

A selection over Table 6.5 $\sigma_{B=b}(R)$ would result in the selections of rows 1 and 2. To include row 3, a new operator '?=' is introduced, which states that for

**Table 6.5. Relation Example**

| R (A | B | C | D) |
|---|---|---|---|
| a | b | c | d |
| g | b | e | $\varphi$ |
| f | $\varphi$ | e | d |
| k | h | $\varphi$ | $\varphi$ |
| p | h | e | $\varphi$ |

two attribute values A and B,

$$A \mathrel{?=} B \qquad \text{iff} \qquad (A = B) \lor (A = \varphi) \lor (B = \varphi) \tag{6.7}$$

Projection produces a new relation which includes only those attributes specified. Since this does not include an evaluation of local nulls against a value, it is possible to include local nulls that exist within those columns into the new relation. However, there is a possibility that the new relation will contain duplicate tuples. For example, a projection, $\pi_{C,D}$ over the relation in Table 6.5 will result in the following relation (Table 6.6). This relation consists of a duplicate tuple

**Table 6.6. Projection Example**

| (C | D) |
|---|---|
| c | d |
| e | $\varphi$ |
| e | d |
| $\varphi$ | $\varphi$ |
| e | $\varphi$ |

<e, $\varphi$ >, which can be deleted as the evaluation of local null against another local null is that they are the same. This is reasonable since including the extra duplicating tuples does not provide any additional information. Similarly tuple, $< \varphi, \varphi >$ does not provide any useful information, and is removed. Additionally, the evaluation requires multiple conditions joined together by logical operators. For example $\pi_{B,C}\sigma_{B?=b \land C?=e}$, produces Table 6.7, in which the first row is true and the second row is locally unknown.

**Table 6.7. Multiple Conditions Example**

| (B | C) |
|---|---|
| b | e |
| $\varphi$ | e |

### 6.3.3 Join Operations

Unlike set theory operations where the relations being used must be compatible, join operations allow the joining of relations that is defined over different schemata. These operations involve the combination of two relations over compatible attributes. A conventional join of two LP-relations produce a new relation whose tuples are true with respect to the conditions of the join.

**Table 6.8. Join Example**

R

| (A | B | C | D) |
|---|---|---|---|
| a | b | c | d |
| g | b | e | $\varphi$ |
| f | $\varphi$ | e | d |
| k | h | $\varphi$ | $\varphi$ |
| p | h | e | $\varphi$ |

S

| (E | F | G | H) |
|---|---|---|---|
| l | a | d | k |
| m | a | h | $\varphi$ |
| n | $\varphi$ | e | d |
| o | $\omega$ | $\varphi$ | $\varphi$ |
| q | k | e | $\varphi$ |

Assuming a conventional join over the attributes $A$ and $F$ of relations $R$ and $S$ in Table 6.8, (i.e., $R \bowtie_{A=F} S$), the relation in Table 6.9 would be produced.

**Table 6.9. Conventional Join**

| (A | B | C | D | E | F | G | H) |
|---|---|---|---|---|---|---|---|
| a | b | c | d | l | a | d | k |
| a | b | c | d | m | a | h | $\varphi$ |
| k | h | $\varphi$ | $\varphi$ | q | k | e | $\varphi$ |

Methods to generalise joins over attributes where global nulls exists had been discussed in the literature (Zaniolo 1984, Codd 1975, Lacroix & Pirotte 1976, Maier 1983). These generalised join states that there may also exist tuples where it is globally unknown whether it is true, in addition to tuples that are true. That is, using the null substitution principle proposed by Codd (1975), it is possible to include tuples where the equivalence evaluation over attribute values that are globally null indicates a globally unknown solution. These tuples are included since they suggests a solution that might be true. For example, Table 6.10 shows a proposed generalised join, denoted $\bowtie^G$, over the attributes $A$ and $F$, $R \bowtie^G_{A=F} S$. For this example, tuples 4 to 8 are global unknown solutions.

In fact, $R \bowtie^{G}_{A=F} S$ can be considered equivalent to a conventional join where $R \bowtie_{(A=F)\cup(A=\omega)\cup(F=\omega)} S$. The local join over the attributes $A$ and $F$ would

**Table 6.10. Generalised Join**

| (A | B | C | D | E | F | G | H) |
|---|---|---|---|---|---|---|---|
| a | b | c | d | l | a | d | k |
| a | b | c | d | m | a | h | $\varphi$ |
| k | h | $\varphi$ | $\varphi$ | q | k | e | $\varphi$ |
| a | b | c | d | o | $\omega$ | $\varphi$ | $\varphi$ |
| g | b | e | $\varphi$ | o | $\omega$ | $\varphi$ | $\varphi$ |
| f | $\varphi$ | e | d | o | $\omega$ | $\varphi$ | $\varphi$ |
| k | h | $\varphi$ | $\varphi$ | o | $\omega$ | $\varphi$ | $\varphi$ |
| p | h | e | $\varphi$ | o | $\omega$ | $\varphi$ | $\varphi$ |

evaluate the true solutions in addition to locally unknown solutions. That is,

$$\begin{aligned} R \bowtie^{L}_{A=F} S &\equiv R \bowtie_{(A?=F)} S \\ &\equiv R \bowtie_{(A=F)\cup(A=\varphi)\cup(F=\varphi)} S \end{aligned} \tag{6.8}$$

This may be seen in Table 6.11. Similarly, a generalised local join is also proposed, denoted $\bowtie^{GL}$. That is,

$$R \bowtie_{(A=F)\cup(A=\omega)\cup(F=\omega)\cup(A=\varphi)\cup(F=\varphi)} S \tag{6.9}$$

**Table 6.11. Local Join**

| (A | B | C | D | E | F | G | H) |
|---|---|---|---|---|---|---|---|
| a | b | c | d | l | a | d | k |
| a | b | c | d | m | a | h | $\varphi$ |
| g | b | e | $\varphi$ | o | g | $\varphi$ | $\varphi$ |
| k | h | $\varphi$ | $\varphi$ | q | k | e | $\varphi$ |
| a | b | c | d | n | $\varphi$ | e | d |
| g | b | e | $\varphi$ | n | $\varphi$ | e | d |
| f | $\varphi$ | e | d | n | $\varphi$ | e | d |
| k | h | $\varphi$ | $\varphi$ | n | $\varphi$ | e | d |
| p | h | e | $\varphi$ | n | $\varphi$ | e | d |

Outer joins are also possible for tables with local nulls. There exist two outer join operations, left outer join and right outer join. Both perform similar operations whereby it includes the tuples which would result from a normal join, in addition to any remaining tuples from one table padded with global null values, since they are unable to be joined with the other table. The difference between the left and the right outer joins is that the tuples from the first or left table in the join are padded in the former, while the tuples in second or right table in the join are padded in the latter. A local outer join, denoted $\bowtie^{LLO}$ and $\bowtie^{RLO}$ for left and right respectively, is then proposed to allow attributes to join over local nulls, but still be padded with global nulls, i.e.,

$$R \bowtie^{LLO} S \equiv R \bowtie_{(A=F)\cup(A=\varphi)} S \tag{6.10}$$

$$R \bowtie^{RLO} S \equiv R \bowtie_{(A=F)\cup(F=\varphi)} S \tag{6.11}$$

A summary of join operations, including both conventional and the proposed local operations, and their corresponding conditions over the attributes $A, B$ is shown in Table 6.12.

**Table 6.12. Summary of Local Join Operations**

| Operation | | Conditions |
|---:|---|---|
| Normal Join | $\bowtie$ | $A = B$ |
| Local Join | $\bowtie^L$ | $A? = B$ |
| Left Outer Join | $\bowtie^{LO}$ | $(A = B) \cup (A = \omega)$ |
| Left Local Outer Join | $\bowtie^{LLO}$ | $(A = B) \cup (A = \omega) \cup (A = \varphi)$ |
| Right Outer Join | $\bowtie^{RO}$ | $(A = B) \cup (B = \omega)$ |
| Right Local Outer Join | $\bowtie^{RLO}$ | $(A = B) \cup (B = \omega) \cup (B = \varphi)$ |
| Generalised Join | $\bowtie^{G}$ | $(A = B) \cup (A = \omega) \cup (B = \omega)$ |
| Generalised Local Join | $\bowtie^{GL}$ | $(A? = B) \cup (A = \omega) \cup (B = \omega)$ |

## 6.3.4 Division Operation

A procedure for the division operation may be found in (Elmasri & Navathe 2000). Table 6.13 shows an example.

**Table 6.13. Division Example - $R \div S = T$**

R

| (A | B) |
|---|---|
| a | b |
| g | b |
| f | $\varphi$ |
| $\varphi$ | h |
| p | h |

S

| (A) |
|---|
| a |
| g |

T

| (B) |
|---|
| b |

When undertaking division on tables where local nulls are involved, maybe true evaluations could also be included. For example, $R \div_L Q$ would produce a result, $T$, which maybe true since there is a possibility that the local null value is the value $b$ (Table 6.14).

**Table 6.14. Local Null Division Example**

R

| (A | B) |
|---|---|
| a | b |
| g | b |
| f | $\varphi$ |
| $\varphi$ | h |
| p | h |

Q

| (A) |
|---|
| a |
| g |
| f |

T

| (B) |
|---|
| b |

## 6.4 Local Nulls and SQL

To take advantage of the notion of locality using local nulls, extensions to the query language are required.For example, it is now possible to extend SQL to enable the querying for tuples that are locally unknown.

```
SELECT  *      FROM R
        WHERE  A = LNULL
```

Where the new reserved word '*LNULL*' represents local null values.

It is also possible to define queries that results in tuples that are true and locally unknown. That is, using the new operator for relational algebra, '? =', a new query statement such as

```
SELECT  *      FROM R
        WHERE  A ?= 'b'
```

may be constructed.

For local joins,

$$\begin{aligned} R \bowtie^{L} S &\equiv R \bowtie_{(A?=B)} S \\ &\equiv \sigma_{(A?=B)}(R \times S) \end{aligned} \tag{6.12}$$

In which case, we may now convert it to

```
SELECT  *      FROM R,S
        WHERE  A ?= B
```

Additionally, it is also possible to convert the '?=' operator further.

$$\begin{aligned} R \bowtie^{L} S &\equiv R \bowtie_{(A=B)\cup(A=\varphi)\cup(B=\varphi)} S \\ &\equiv \sigma_{(A=B)\cup(A=\varphi)\cup(B=\varphi)}(R \times S) \end{aligned} \tag{6.13}$$

which in SQL is equivalent to

```
SELECT  *       FROM R,S
        WHERE   A = B
        OR      A = LNULL
        OR      B = LNULL
```

This now provides a way to separate local unknown solutions from true solutions. Similarly, for generalised local joins,

$$\begin{aligned} R \bowtie^{GL} S &\equiv R \bowtie_{(A=B)\cup(A=\omega)\cup(B=\omega)\cup(A=\varphi)\cup(B=\varphi)} S \\ &\equiv \sigma_{(A=B)\cup(A=\omega)\cup(B=\omega)\cup(A=\varphi)\cup(B=\varphi)}(R \times S) \end{aligned}$$

which in SQL is equivalent to

```
SELECT  *       FROM R,S
        WHERE   A = B
        OR      A = ω
        OR      B = ω
        OR      A = φ
        OR      B = φ
```

For the other local operations, a conversion using the equivalent condition algebra as shown in Table 6.12 is possible and Table 6.15 provides a summary.

**Table 6.15. Summary of Local Join Operations in SQL**

| Operations | | SQL |
|---|---|---|
| Local Join | $R \bowtie^{L} S$ | SELECT * FROM R,S<br>WHERE A?=B |
| Left Local Outer Join | $R \bowtie^{LLO} S$ | SELECT * FROM R LEFT LOCAL JOIN S<br>WHERE A=B |
| Right Local Outer Join | $R \bowtie^{RLO} S$ | SELECT * FROM R RIGHT LOCAL JOIN S<br>WHERE A=B |
| Generalised Local Join | $R \bowtie^{GL} S$ | SELECT * FROM R,S<br>WHERE A=B OR A=$\varphi$ OR B=$\varphi$<br>OR A=$\omega$ OR B=$\omega$ |

For the example, assuming that the medical practitioner decides to looks up the address of a patient. The following query can now be submitted:

```
SELECT  NAME,ADDRESS,AGE,TELEPHONE
        FROM   PATIENT
        WHERE  NAME ?= 'John Doe'
        AND    AGE ?= 30
```

which would result in the Table 6.16.

**Table 6.16. Example Results**

| **ID** | Name | Address | Age | Telephone |
|---|---|---|---|---|
| 1 | John Doe | 22 Ever St... | 30 | NULL |
| 2 | LNULL | LNULL | 30 | LNULL |

In our running example, the practitioner may be satisfied with the result of ID 1 if enough information is already displayed. However, since the 'Telephone' attribute of ID 1 returns a NULL value, no additional query is required from the main database. If the telephone attribute of ID 2 is required, it can be obtained through further querying of the main database. The sub-query transmitted to main database may then more specific and thus, reduce the time and communication bandwidth required.

It is interesting to note that a local null appearing in the `SELECT` clause of an SQL statement is generally less of an issue that one appearing in a `WHERE` clause. For example, given the SQL example above, a local null in `AGE` would mean that the selection of the complete tuple is conditional, whereas a local null in `ADDRESS` would merely mean that the value is unknown but the presence of the tuple is unconditional.

## 6.5 Discussion

In this chapter, the concept of local nulls in relations was introduced, where it was shown that it is possible to manipulate relations with these nulls through the use of extended relational algebra. It is then possible to extend the SQL language to identify locally unknown solutions and, importantly, to provide users with a definable response to queries that include them.

The concept of local nulls are primarily applicable within a distributed or mobile database system, where a piece of information exists in multiple databases within that system. Because local nulls are considered to be temporary nulls that replace actual values within a database in order to conserve storage capacity, it is possible for attribute values to be nulls, in a local database, while other databases can have its actual value. In mobile databases, where the storage capacity is limited, it is useful to have the ability to store only parts of the database that are used often. For values that are not stored, local nulls are used to allow a database to determine whether the information being sought by the can be found in other databases within its system.

# Chapter 7

# Transaction Management

For a mobile distributed system to use COSMOS in a hierarchical architecture, current transaction processing concepts must be examined, and if necessary, modified. In traditional distributed transactions, the ACID properties must be satisfied. However, due to the issues within the mobile environment (see Section 1.2), mobile transactions will not be able to adhere to strict ACID properties. Modification to the ACID properties are therefore required.

This chapter is organised as follows. In section 7.1 a transaction model is presented, which examines the relaxation of certain ACID properties. In particular, the relaxation of consistency allowing mobile transactions to occur, is discussed in Section 7.2. A reconciliation scheme to ensure consistency is then discussed in Section 7.3. A discussion on this chapter is presented in Section 7.4.

## 7.1 Transaction Model

Within a hierarchical centralised distributed system, there is one authoritative copy of the database, the Central Database (CDB). All other sites will either have a local or server Summary Database (SDB), depending upon whether it is capable of being a server site. Sites that have a direct connection to a server SDB are said to *exist on a same branch* as the server.

Within the system, sites can be weakly or strongly connected to other sites. Strong connectivity refers to connections that are attained through high-bandwidth and low-latency communications, while weak connectivity refers to as those with intermittent or low bandwidth (Pitoura & Bhargava 1999, Madria & Bhargava

1999). Weak connectivity is expensive in terms of cost and speed, and is unreliable. Two strongly connected sites implies that both are reliably connected (strong connectivity) and disconnections only occur through failure. Alternatively, two weakly connected sites exhibit frequent disconnections, whether voluntary or not. Additionally, widely distributed database systems display similar network connectivity conditions. A 'strong branch' includes sites in the same branch that have strong connectivity. Thus, in order to reduce the amount of communication over weakly connected sites, local and global transactions are introduced that access summarised copies of the database.

In addition, traditional distributed transactions (strict transactions), are supported that are atomic, consistent, durable and isolated, satisfying the ACID properties. That is, the transaction is:

- atomic when all of its operations are executed or none at all,
- consistent if completing its execution will maintain the database consistency,
- isolated, in that the transaction does not view the partial results of any other transactions, and
- durable if any changes applied to the database are permanent after committing the transaction.

For mobile transactions, atomicity is a restrictive property since they are prone to error and can be long-lived. Thus, using the structure (described in Sections 7.1.1 and 7.2) a transaction is divided into subtransactions, each of which must be atomic. The consistency requirement is also restrictive when used for mobile transactions. That is, out-of-date local data can be read when the mobile database is disconnected. Thus, a more relaxed consistency requirement is required and is discussed in Sections 7.1.2 and 7.2. The isolation property is application-specific (Pitoura & Bhargava 1994), and relates to the sharing of results between concurrently executing transactions. This is desirable for different types applications and is not affected by mobility. Due to the error-prone nature of mobile transactions, it is difficult to enforce durability (see Section 7.1.3). Finally, mobile transactions can also support a new property (transaction relocation) examined in Section 7.1.4.

### 7.1.1 Transaction Structure

In order to increase availability and provide support for local nulls, local operations are defined such that they provide local processing of transactions using the SDB. The two local SDB operations are Local Read (LR) and Local Write (LW).

A second set of operations, Global Read (GR) and Global Write (GW) attempt to access server copies of the database. It is assumed that local databases contain subsets of a server, resulting in times when the local database does not contain all the required information. In which case, global operations are used to access the server's database copy.

The traditional operations, renamed from just Read and Write to Strict Read (SR) and Strict Write (SW), respectively, are also supported. These operations attempt to access the data from the central database, and are assumed to be explicitly requested by the user.

To process transaction operations, the database management system translates the operations on the attribute values into operations on the replicated copies of the attribute values. This translation is formalised by a function $h$ (detailed in Section 8.1).

Using these operations, it is now possible to define a transaction as a partial ordering of operations. That is, following the definitions by Bernstein et al. (1987), a transaction is a partial order $(op, <)$, where $op$ is the set of operations executed by the transaction, and $<$ is the execution order. Where operations include the data operations, Local Read (LR), Local Write (LW), Global Read (GR), Global Write (GW), Strict Read (SR), Strict Write (SW), Abort (A) and Commit (C). The partial order specifies the execution order the read and write operations and contain either an abort or commit at the end. Two data operations conflict if they access the same copy of an attribute value and at least one of them is a type of write operation.

To model transaction processing at an intermediate level of the system architecture, GR and GW can be downgraded to LR and LW, respectively. That is, global operations, from a mobile database, will be propagated to its respective server, which then re-evaluates the operations through transaction decomposition to refine the operations. However, if a global transaction is propagated to the CDB the transaction is upgraded to a strict transaction, allowing it to access data within the CDB.

### 7.1.2 Data Consistency

A database state is a snapshot of the attribute values at a particular point in time. To ensure that a database state is correct and consistent, integrity constraints on the attribute values are used (Papadimitriou 1986). A type of integrity constraint is coherence, which defines the restriction between actual values of same attribute values that exist in different databases. For traditional distributed databases, consistency is assumed for fully connected sites. However, this is not true for mobile sites, where intermittent connections are possible. As a result, there is a need to relax consistency constraints and allow variable consistency to exist between databases (Pitoura & Bhargava 1999, Terry et al. 1995).

In the case of COSMOS, any consistency other than full consistency is regarded as weakly consistent. Local integrity constraints are then the integrity constraints, excluding coherency of an attribute value that exists in both the local SDB and UDB. On the other hand, global integrity constraints specify the integrity constraints, including that of coherency, that exist between the same attribute value in different databases. For example, integrity constraints that exist between a local SDB and a global SDB or a local SDB and the CDB.

Thus, a SDB state is consistent when the local integrity constraints are satisfied and the global integrity constraints are bounded-inconsistent, i.e., weakly consistent, when weakly connected and fully consistent otherwise. Note that the local coherence constraint between the SDB and UDB is not required, since UDB only stores local updates until a database refresh is initiated.

Bounded inconsistency implies that when a local database site is weakly connected to its server or central site, there is a bounded divergence between the copy that it holds and the server or central copy (Alonso, Barbara & Garcia-Molina 1990, Sheth & Rusinkiewicz 1990, Pitoura & Bhargava 1999). The degree of divergence, $d$, quantifies the bounded divergence, and the term $d$-consistent indicates how a replication constraint for an attribute value is bounded. The possible definitions of $d$ are:

- The maximum number of transactions that operate on local SDB copies.
- The range of acceptable values that an attribute value may take.
- The maximum number of attribute values that have divergent copies.

The degree of divergence can be changed to suit the current connectivity status.

That is, for strong connectivity, a smaller divergence is required and for weak connectivity, a greater divergence is acceptable.

### 7.1.3 Recovery

Mobile hosts are more inclined to failures than static hosts. Due to their mobility, these hosts can be lost, stolen or damaged. Additionally, their reliance on wireless communications can cause more communication failures (described in Section 1.2) when compared to wired communications.

Recovery protocols ensure the durability of a transaction. However, it is more difficult to ensure the durability of transactions in a mobile environment than in a non-mobile environment, as any mobile operations must be reported to a fixed network to ensure durability.

Traditional database recovery includes several techniques for non-catastrophic transaction failures (Bernstein et al. 1987). These include UN-DO/REDO, UNDO/NO-REDO, NO-UNDO/REDO, and NO-UNDO/NO-REDO algorithms. These algorithms specify whether the recovery technique is able to undo any operations, and whether it is required to redo any operations if a system fails.

For traditional distributed systems this is more complex, especially to maintain atomicity, requiring a two-phase commit protocol (Bernstein et al. 1987). This protocol employs a coordinator to maintain recovery information, such as update logs, and coordinates commit operations on all involved databases.

These recovery techniques are not always suitable for mobile transactions due to the host's susceptibility to failure. Pitoura and Bhargava (1995a), proposed an agent-based model from which mobile transactions are processed in order to access heterogeneous mobile databases. Agents are middleware, situated in a fixed network, through which mobile transactions may be submitted for processing, and therefore extends the client/server architecture into client/agent/server architecture. That is, mobile transactions can be submitted either in parallel to the local database and the agent, or just to the agent. Since the agent is located at a fixed network, it processes the transaction using any available database. If during this time the mobile client fails, either through voluntary or non-voluntary failure, the agent continues processing. Assuming that the agent has a stable storage and can survive failures, durability is ensured even after commit, as the agent still has a copy of any transaction made by the client.

In the case of COSMOS, it is necessary to assume that the CDB has stable storage and is able to survive failures, so that the durability is ensured for strict transactions. For local transactions, all updates are propagated after a local commit. If the required updates are propagated to its server before a failure occurs, then transaction durability is guaranteed for this site. However, since updates require propagation until the CDB is reached, there is still a possibility of failure during this further propagation. Thus, durability for local transactions are only guaranteed if the local transaction reaches the CDB. At this time, it is considered a strict transaction, if it does not violate the serialisability rule as described later in Section 7.3.

### 7.1.4 Transaction Relocation

The idea of mobility is that users move geographically, and changes locations frequently. It is possible that after moving the communication cost to their support station is expensive in terms of communication times. This may result in users moving from their current support station to another of less communication cost. It is then necessary to relocate part of a mobile transaction to the new support station improving response time by using the cheaper communication link.

Dunham et al. (1997) proposed the *Kangaroo Transaction* as a mobile transaction model to capture the movement of a mobile user. That is, a transaction is first divided into different parts, and as the user moves from one support station to another, different parts are processed at different support stations. These transaction parts are then committed and/or aborted independently.

At this point, COSMOS does not provide support for transaction relocation. However, the architecture does not require the relocation of any transaction part, as all strict transactions that affect global commitment are required to be propagated to, and executed at, the CDB. The effect of mobile hosts connecting to a server other than its own requires that its server, and all its supersets, include all attribute values that exist in the SDB. This can be costly if there are no intersections in attribute values between the local and server SDB, as all the local attribute values must be added to the server's database.

## 7.2 Consistency in Local Operations

For the correct execution of concurrent transactions at a local database, only conflicts between local and global operations must be examined. That is, conflicts may occur when users that are local to the server database submits local transactions while users local to any subset of that server database submits global transaction to that server database, concurrently.

For simplicity, if the CDB is isolated by having only weak connectivity to the SDBs, it can be assumed that strict transactions are only executed at the CDB. This assumption, however, may be removed in which case strong consistency is required of those sites and the CDB.

Global transactions are, seen to be, a special case of local transactions. That is, using the hierarchical architecture, global transactions only affect the server site that they are submitted to and all other subsequent server sites to which the global transactions are submitted on the same branch. Other sites on different branches considers those global transactions to be local transactions, and as a result, does not affect those sites.

Thus, a complete Local Schedule (LS) is an observation of an interleaved execution of transactions at the top site of a strong branch. Let $T = \{T_0, T_1, ..., T_n\}$ be a set of transactions. A complete Local Schedule (LS) over $T$ is formally, a pair ($op$, $<_L$) in which $<_L$ is a partial ordering relation such that (Bernstein et al. 1987):

1. $op = h(\bigcup_{i=0}^{n} T_i)$ for some translation function $h$.
2. For each $T_i$ and all operations $p_i$,$q_i$ in $T_i$, if $p_i <_i q_i$, then every operation in $h(p_i)$ is related by $<_L$ to every operation in $h(q_i)$.
3. For all read operations, $r_j[x_i]$, there is at least one write operation, $w_k[x_i]$, such that $w_k[x_i] <_L r_j[x_i]$ and $w_k[x_i] \in SW_k[x]$.
4. All pairs of conflicting operations are related by $<_L$, where two operations conflict if they access the same copy of an attribute value and one of them is a write operation.
5. If $w_j[x] <_L r_j[x]$ and $r_j[x_i] \in h(r_j[x])$ then $w_j[x_i] \in h(w_j[x])$.

Condition 1 states that the database system translates each operation on an attribute value into the appropriate operations on the data copies. Condition 2 states that the local schedule preserves the ordering stipulated by each transaction $T_i$. Condition 3 states that a transaction is not allowed to read a copy unless it has been previously initialised in the central database. Condition 4 states that the local schedule records the execution order of conflicting operations. Condition 5 states that, if a transaction writes into an attribute value $x$ before it reads $x$, then it must write to the same copy of $x$ that it will subsequently read.

For a set of transactions, $T$, a read operation on an attribute value $x$ *reads-x-from* a transaction $T_i$, if it reads a copy of $x$ written by $T_i$ and no other transaction writes to this copy of $x$ in between. A transaction $T_i$, then has the same *read-from* relationship in two schedule, $S_1$ and $S_2$, if $T_i$ *reads-x-from* $T_j$ in $S_1$ and it *reads-x-from* $T_j$ in $S_2$.

Two schedules are *conflict equivalent* if for the same set of transactions, the order of any two conflicting operations are the same in both schedules (Bernstein et al. 1987). A *one-copy* schedule is a single-copy interpretation where all operations on data copies are represented as operations on the corresponding attribute value (Pitoura & Bhargava 1999). A schedule is then *one-copy* serialisable if it is equivalent to a serial one-copy schedule.

### 7.2.1 Correctness Criterion

Bounded inconsistency must be maintained for the correct concurrent execution of local, global and strict transactions. The correctness criterion for strict transaction execution is one-copy serialisability at the CDB. One-copy serialisability ensures that all strict transactions have a consistent view as a one-copy serial schedule. This occurs since strict transactions only exist in the CDB and thus, consistency is required to only be managed centrally.

For global and local transactions, the correctness criterion is a weaker form of *one-copy* serialisability. That is, only serialisability is required between the schedule of the top server site of a branch, and all other branch sites. As a result, the sites are only required to read consistent data that has been globally or branch committed, by either strict transactions or global transactions from the same branch, respectively. Branch commit refers to a global transaction being committed at the top branch site.

Two schedules $S$ and $S'$ are said to be *view equivalent* if the following three conditions hold (Bernstein et al. 1987):

1. For a set of transactions $T$, all operations in $T$ exist in both $S$ and $S'$.

2. Both $S$ and $S'$ has the same read-from relationship.

3. Each final set of writes in $S$ is the same as the final set of writes in $S'$

So a Local Schedule (LS) is weakly correct if and only if the following conditions are true:

1. All transactions in LS have a consistent view.

2. There exists a one-copy serial schedule, $S$, such that $S$ is view equivalent to a strict projection of LS.

3. There is bounded-inconsistency between weakly connected sites.

Given that bound-inconsistency exists, it is possible to show that LS is weakly correct in terms of equivalence to a serial schedule. That is,

1. a projection over the strict transactions of LS is one-copy serialisable, and

2. a projection on a site is conflict equivalent to a serial schedule.

Thus, assuming bounded inconsistency between weakly connected sites, it is only necessary to guarantee a consistent view for transactions that occur between sites that are strongly connected. Assuming that there exists a serial schedule for every transaction, then projecting that schedule to only include transactions that affect a site will also provide a serial schedule. For local transactions, since they only operate locally, it is possible to assume a consistent view if a projection of each transaction on the site satisfies local integrity constraints when executed alone. Similarly for global transactions, only local integrity constraints at the top branch site, within a strong branch, must be satisfied.

Thus, for strict transactions one-copy serialisability ensures that all strict transactions have a consistent view as a one-copy serial schedule. For local and global transactions, it suffices to have serialisability for the projection of the schedule on the top branch site.

This serialisability ensures that local and global transactions from strongly connected sites, within the same branch, will have a consistent view of strict transactions. For sites that are weakly connected, the correctness criterion does not ensure that local and global transactions have the same view. Thus, a stronger definition for correctness criterion is proposed, which is similar to that defined by Pitoura and Bhargava (1999):

A Local Schedule (LS) is strongly correct if and only if there is a serial schedule $S_S$

1. $S_S$ is conflict equivalent with LS.
2. There exists a one-copy serial schedule, $S$, such that $S$ is view equivalent to a strict projection of $S_S$.
3. There are bounded-inconsistencies between weakly connected sites.

Strong correctness then requires synchronisation of local and global transactions regardless of where they originate from. Alternatively, weak correctness only requires that the top site of a branch keeps track of any local and global transactions that affect it. Therefore, no concurrency control messages would be required over weak connections, since strict transactions are executed and committed on the CDB, and local and global transactions are synchronised within strong branches.

### 7.2.2 Serialisation Graph

The correct execution of a LS can be determined by extending the serialisation graph, based on methods proposed by Pitoura and Bhargava (1999). In which, a serialisation graph is first constructed using all strict transactions, then conflicting local transactions are included. Global transactions are disregarded in the serialisation graph and are considered as local transactions of the server site to which they were submitted. This is because a global transaction is only global at the originating site and at all its subsequent subsets or SDB. The conflicts between local and strict transactions are usually due to local reads and strict writes. Thus, the following edges exist for a Local Serialisation Graph (LSG) between local and strict transactions induced by a LS:

1. Dependency edge from $ST$ to $LT_i$.
2. Precedence edge from $LT_i$ to $ST$.

Where $ST$ is a strict transaction and $LT_i$ is a local transaction from site $i$. A dependency edge then represents the fact that a transaction reads a value that has been changed by another transaction. While a precedence edge represents a transaction reading a value that was later changed by another transaction.

For a LS to be strongly correct, the LSG must be acyclic. It has been shown (Pitoura & Bhargava 1999) that when acyclic, each of its subgraphs is acyclic, and thus the replicated data serialisation graph on which the LS is built on is acyclic. The proof by Pitoura and Bhargava (1999) is presented in the context of local transactions:

> Acyclicity of the replicated data serialisation graph implies one-copy serialisability of strict transactions, since strict transactions only read values produced by strict transactions. Let $T_1, T_2, ..., T_n$ be all local and strict transactions in LS. Thus, $T_1, T_2, ..., T_n$ are the sites of the LS. Since LS is acyclic it can be topologically sorted. Let $T_{i_1}, T_{i_2}, ..., T_{i_n}$ be a topological sort of the edges in LS, then by a straightforward application of the serialisability theorem (Bernstein et al. 1987), LSG is conflict equivalent to the serial schedule $S_S = T_{i_1}, T_{i_2}, ..., T_{i_n}$. This order is consistent with the partial order induced by a topological sorting of the replicated data serialisation graph, let $S_{1C}$ be the one-copy serial schedule corresponding to this sorting. Thus, the order of transactions in $S_S$ is consistent with the order of transactions in $S_{1C}$.

## 7.3 Consistency Reconciliation

After the execution of several local and strict transactions, bounded inconsistency will occur where, for each attribute value, there will be differences in value between the same attribute values in CDB and all its subsequent SDBs. There are many approaches that for reconciling the different attribute values into one, ranging from purely syntactic to purely semantic (Davidson et al. 1985). Syntactic approaches use serialisability-based criteria to restore the database state. While, semantic approaches use the semantics of either the transactions or attribute values to reconcile. For this thesis's purposes, a purely syntactic approach has been selected, based on methods by Pitoura and Bhargava (1999), since it is application independent. Reconciliation occurs at different times at different SDBs. For example, it can occur to keep the consistency of a SDB bounded, or occur periodically or when strong connectivity is achieved between two databases.

### 7.3.1 Correctness Criterion

The rule of reconciliation requires that a local transaction, which subsequently propagates to high tier sites as global transaction, becomes globally committed if the inclusion of its write operations does not violate the one-copy serialisability of strict transactions (Pitoura & Bhargava 1999). In the case of COSMOS, reconciliation occurs at two places, a CDB or a server SDB. Therefore, global transactions from different branches leading to a CDB must not interfere with each other and any strict transactions. While, global transactions from different lower tier branches leading to a server SDB must not interfere with each other or any local transaction originating from the server. A complete Global Schedule (GS) models execution after reconciliation when the strict transaction, or local transaction at the server SDB, becomes aware of the global transaction. Thus, relevant conflicts between global and strict, or global and local, operations are reported in addition to the conflicts already reported by LS.

A Global Schedule (GS) is then a pair $(op',<_G)$ based on the Local Schedule (LS), where

1. $op' = G(op)$.
2. For any $p_i$ and $q_i \in op'$, if $p_i <_L q_i$ in $G$(LS) then $p_i <_G q_i$ in GS.
3. For each pair of global write $p_i = GW_i[x]$ and strict read $q_j = SR_j[x]$ operations, all pairs of operations $l_i \in h(p_i)$ and $s_j \in h(q_j)$ where either $l_i <_G s_j$ or $s_j < l_i$.
4. For each pair of global write $p_i = GW_i[x]$ and strict write $q_j = SW_j[x]$ operations, all pairs of operations $l_i \in h(p_i)$ and $s_j \in h(q_j)$ where either $l_i <_G s_j$ or $s_j < l_i$.

Condition 1 and 2 specify that the GS contains the same operations and ordering as a LS, however, all local operations are converted using some function, $G$, to global operations. Condition 3 and 4 then define extensions where strict transactions, or local if reconciliation is occurring at a server SDB, have read-from relationships with local transactions. That is, a strict operation on an attribute value $x$ *reads-x-from* a transaction $T_i$ in a GS, if it reads a copy of $x$ from which $T_i$ has written to in the CDB or any subsequent SDB and no other transactions write any copy of $x$ in between.

The correctness of a Global Schedule (GS) is then true if and only if:

1. it is based on a correct LS, and
2. the reads-from relationship for strict transactions are the same with their reads-from relationships in the LS.

Similarly, a reconciliation occurring at a server SDB is defined using the same definition as that which occurs at the CDB, but substituting all strict operations to local operations originating at the server SDB. Since a server SDB can still have bounded inconsistency to the CDB, only transactions that originate from its branch are required for reconciliation purposes.

### 7.3.2 Serialisation Graph

Using methods described in (Pitoura & Bhargava 1999), a modified serialisation graph, called Global Schedule Graph (GSG), is used to determine the correctness of GS. That is, the graph of a LS, LSG, is used as a base to construct a GSG. Note that only reconciliation at the CDB is considered here, since any reconciliation occurring at any server SDB uses the same notation, once all strict transactions are converted to local transactions of that site.

To illustrate the conflict between weak and strict transactions that access different copies of the same attribute value, the following edges are induced for:

1. a write order, if $T_i$ local writes and $T_k$ strict writes any copy of an item $x$ then either there is an edge $T_i \rightarrow T_k$ or $T_k \rightarrow T_i$, and
2. a strict read order, if a strict transaction $ST_j$ *reads-x-from* $ST_i$ in LS and a global transaction LT follows $ST_i$, then an edge $ST_j \rightarrow GT$ is added.

Thus, a GS is correct if the GSG for the GS is acyclic, as proven Pitoura and Bhargava (1999), and is extended as follows:

> Clearly, if the GSG is acyclic, the corresponding graph for the LS is acyclic (since to get the GSG we only add edges to the LSG) and thus the LS is correct. To show that if the graph is acyclic, then the reads-from relationship for strict transactions in the GS is the same as in the underlying LS. Assume that $ST_j$ *reads-x-from* $ST_i$ in LS. Then $ST_i \rightarrow ST_j$. Assume for the purposes of contradiction, that $ST_j$ *reads-x-from* a global transaction $GT$. Then $GT$ writes $x$ in GS and

since $ST_i$ also writes $x$ either (a) $ST_i \rightarrow GT$ or (b) $GT \rightarrow ST_i$. In case (a), from the definition of GS, there is an edge $ST_j \rightarrow GT$, which is a contradiction since $ST_j$ *reads-x-from* $GT$. In case (b), $WT \rightarrow ST_i$, that is $GT$ precedes $ST_i$ which precedes $ST_j$, which again contradicts the assumption that $ST_j$ *reads-x-from* $GT$.

## 7.4 Discussion

The management of transactions discussed in this chapter is based on allowing these transactions to work within different types of communication environments. As such, the ACID requirements for transactions work well for traditional distributed databases, where the communications used are relatively high in bandwidth. However, due to the resource issues of operating within a mobile environment, relaxation of certain ACID properties are required. In particular, this chapter explored the relaxation of the consistency requirements and the subsequent reconciliation required, in terms of the serialisability of schedules. The method proposed is based on previous work by Pitoura and Bhargava (1999). A similarity between their method and the method presented in this chapter is the support for weak connectivity operations and the syntactic approach to reconciling data between different databases.

While similarities exists, there are also many differences. In particular, COSMOS focuses on a hierarchical database architecture. This leads to the definition of three types of transactions, local, global and strict transactions. Where a local transaction operates only at an SDB, a global transaction captures the database access of a user attempting to request data that are not available locally. Finally, a strict transaction allows a user to access up-to-date information stored at the CDB.

Another difference occurs during reconciliation, which can now occur at different places, a server SDB or CDB. This requires identifying conflicts between global and local transactions when occurring at a server SDB, or between global and strict transactions when occurring at the CDB. As a result aborts may now occur before reconciliation reaches the CDB.

In practice, the testing of the serialisability of a schedule is quite difficult especially when a SDB has no access to the transactions of other SDB's from different branches. Thus, protocols are developed to ensure the serialisability of all schedules. The next chapter discusses different protocols in detail, including the ones that ensure serialisability.

# Chapter 8

# Transaction and System Failure Protocols

Protocols ensure that a mobile distributed database system runs as expected, by employing a number of different protocols that are responsible for different system aspects. This chapter examines protocols for transactions and system failures. Transaction management protocols deal with how transactions are handled within the system. This includes the locking protocols used by the system to deal with how concurrent transactions access the same data, and update mechanisms that deal with data updates. The system management protocols examined in this chapter deal primarily with how each site may react if the CDB is unavailable for a long period of time.

This chapter will be organised as follows: Section 8.1 discusses the transformation function requirement to separate a transaction into local and global subtransactions. Sections 8.2 and 8.3 examine concurrency control and reconciliation mechanisms, respectively, to ensure serialisability as discussed in the Chapter 7. Sections 8.4 and 8.5 examines local update mechanisms. Section 8.6 examines the possible protocols to bound inconsistencies between sites. Section 8.7 examines database refresh procedures required to propagate CDB updates. Finally, Section 8.8 discusses the recovery techniques required, if the CDB is unavailable to any SDB, due to failure or network partitioning.

## 8.1 Transformation Function

As Storage Map (SM) provide a method to describe the contents of the SDB, a query transformation, $h$, is required to construct a corresponding storage map.

There are two main methods used for $h$ both of which utilise SMs during the transformation.

The first is for read-only transactions (queries). To determine which part of a read-only transaction may be processed by the SDB, new a SM is created for the transaction. As an example, consider a map description for the relation, Patient, in Figure 5.3 (see Table 8.1). In the figure the Patient Identifier, patCode, is included in full, while other attributes are listed as boolean values indicating that the value is held in the SDB.

Given that the following query, $Q1$, is part of the submitted read-only transaction, (Table 8.1),

```
SELECT  Name   FROM Patient
        WHERE  SEX = 'F'
        AND    patCode < 1003
```

Assuming that this query then induce the following read operations on the relation, Patient:

$r[(1000, name)]r[(1001, name)]r[(1002, name)]$.

These read operations directly map onto an SM (see Table 8.1). The easiest way to determine if a query can be answered is to perform a bitwise AND NOT operation to the description. That is, if $Q \wedge \neg S = 0$ where $Q$ and $S$ are the storage maps of the query (Q1) and the summary database (Patient), respectively, then the query is answerable. In addition, a non-zero answer also indicates which particular items are causing the inability to answer.

As can be seen from Table 8.1, $R1 \neq 0$ and thus the above query is unanswerable within the summarised Patient relation. However, $R1$ now provides the missing attribute values that can be used as a request to the server. That is, global operations may be induced from $R1$ to create a global transaction. In this case, $GR[(1000, name)]GR[(1001, sex)]GR[(1002, sex)]$.

Meanwhile, as the global transaction is created for submission to the server (if available), the local SDB may also run the query and return local nulls for those attribute values that are not available locally. This can be achieved through the use of local null SQL rewrites as described in Chapter 6, which is application dependent and may also depend on the accuracy required by the user. This procedure is also used when the local SDB is disconnected from the server. Assuming the user does not require high accuracy, the SQL statement for $Q1$ may then be rewritten as:

**Table 8.1: Storage Map Examples for** *Patient* **Relation, Query,** $Q1$ **and Result** $R1 = (Q1 \wedge \neg Patient)$

| **Patient** | *patCode* | **name** | **sex** | **age** | **town** | **physician** |
|---|---|---|---|---|---|---|
| | *1000* | 0 | 1 | 0 | 1 | 0 |
| | *1001* | 1 | 0 | 0 | 0 | 1 |
| | *1002* | 1 | 0 | 0 | 1 | 1 |
| | *1003* | 0 | 0 | 0 | 0 | 1 |
| | *1004* | 0 | 1 | 1 | 0 | 1 |
| | *1005* | 1 | 1 | 0 | 0 | 0 |
| | *1006* | 1 | 1 | 1 | 1 | 1 |
| | *1007* | 1 | 1 | 0 | 0 | 1 |

| **Q1** | *patCode* | **name** | **sex** | **age** | **town** | **physician** |
|---|---|---|---|---|---|---|
| | *1000* | 1 | 1 | 0 | 0 | 0 |
| | *1001* | 1 | 1 | 0 | 0 | 0 |
| | *1002* | 1 | 1 | 0 | 0 | 0 |
| | *1003* | 0 | 0 | 0 | 0 | 0 |
| | *1004* | 0 | 0 | 0 | 0 | 0 |
| | *1005* | 0 | 0 | 0 | 0 | 0 |
| | *1006* | 0 | 0 | 0 | 0 | 0 |
| | *1007* | 0 | 0 | 0 | 0 | 0 |

| **R1** | *patCode* | **name** | **sex** | **age** | **town** | **physician** |
|---|---|---|---|---|---|---|
| | *1000* | 1 | 0 | 0 | 0 | 0 |
| | *1001* | 0 | 1 | 0 | 0 | 0 |
| | *1002* | 0 | 1 | 0 | 0 | 0 |
| | *1003* | 0 | 0 | 0 | 0 | 0 |
| | *1004* | 0 | 0 | 0 | 0 | 0 |
| | *1005* | 0 | 0 | 0 | 0 | 0 |
| | *1006* | 0 | 0 | 0 | 0 | 0 |
| | *1007* | 0 | 0 | 0 | 0 | 0 |

```
SELECT  Name   FROM Patient
        WHERE  patCode < 1003
        AND    SEX = 'F'
        OR     SEX = 'LNULL'
```

The second transformation function is required for read-write transactions. To determine the required database, a SM conversion is required to determine the local availability of the attribute values. This conversion is the same procedure as that described in the first method. If all the required attribute values exist in the local SDB then the update id performed locally. However, if there exists a required attribute value that is a local null, dependencies must exist between the operation on the locally unknown value and other operations, the entire transaction becomes a global transaction and is propagated to the server.

## 8.2 Serialisability

For concurrency control it is possible to use a strict two phase locking protocol, for which a transaction is required to release all locks on commit. This is similar to the two level serializability scheme used in multidatabases (Breitbart, Garcia-Molina & Silberschatz 1992, Breitbart, Garcia-Molina & Silberschatz 1995). For a strict transaction, locks are released after global commitment. That is, the transaction is committed at the CDB while all databases maintain bounded inconsistency. For local transactions, locks are released after local commit, while for global transactions, locks are released after local commitment at the server database.

There are thus six lock modes (LR,LW,GR,GW,SR,SW) that correspond to the different data operations. A lock is required before the execution of an operation, and is granted when the attribute value is not currently locked in an incompatible lock. Locks for strict, global and local transactions are processed at the CDB, server SDB and local SDB, respectively. Lock compatibility between different lock are shown in Table 8.2 and 8.3.

Propagation of strict updates from the CDB are asynchronous, or lazy, using database refresh procedures. This occurs since the CDB keeps track of all locks for strict transactions (see Section 8.7). There are two methods of lock invocation on an attribute value. These methods reflect whether locks may be granted while a write lock is being held by a global or strict transaction.

The first method does not impose a lock on any site but the top branch server or CDB, when either a global write or strict write is respectively requested. A lock request is then granted for local or local and global transactions when global write or strict write locks are being respectively held. These locks are only granted to sites at which the locks are not currently being held. In the case of disconnections, the second method does not incur any blocking since the locks of strict and global transactions do not interfere with local transactions. However, the database must assess bounded-inconsistency as if those transactions had been committed. The locking matrix for this method is presented in Table 8.2.

**Table 8.2. Locking Matrix 1**

| | | **Held** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **LR** | **LW** | **GR** | **GW** | **SR** | **SW** |
| **Request** | **LR** | Y | N | Y | Y | Y | Y |
| | **LW** | N | N | Y | Y | Y | Y |
| | **GR** | Y | Y | Y | N | Y | Y |
| | **GW** | Y | Y | N | N | Y | Y |
| | **SR** | Y | Y | Y | Y | Y | N |
| | **SW** | Y | Y | Y | Y | N | N |

The second method requires that all global and strict locks are requested from sites that are respectively either of the same branch as the server or globally from all sites. This is achieved by using an 'invisibility mask' on the attribute values, similar to those described in Section 8.5. Thus, all locked items is invisible to all affected sites. During this time, a locking matrix would be induced, as shown in Table 8.3. Incompatibility between lock requests and those held are marked with an 'N' to indicate the lock requests being denied. Thus, no other transactions are possible when a strict lock is being held, and no other local or global transactions are possible when a global lock is held. This, however, has a detrimental effect on the availability of data at the local SDB, as blocking can occur between any two combinations of strict, global and local transactions, when they access the same data. In the case of disconnections, any local transaction will not be able to access any data that has been 'masked'.

**Table 8.3. Locking Matrix 2**

| | | **Held** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **LR** | **LW** | **GR** | **GW** | **SR** | **SW** |
| **Request** | **LR** | Y | N | Y | N | Y | N |
| | **LW** | N | N | Y | N | Y | N |
| | **GR** | Y | Y | Y | N | Y | N |
| | **GW** | N | N | N | N | Y | N |
| | **SR** | Y | Y | Y | Y | Y | N |
| | **SW** | N | N | N | N | N | N |

## 8.3 Reconciliation

Attribute values between different databases are bounded by some degree of divergence and reconciliation is required when strict transactions are globally committed. Reconciliation occurs when a local transaction is propagated to the CDB or server SDB. When a global transaction is submitted to the CDB a correct schedule requires that any potential cycles in the GS graph (GSG) be broken. The construction of a GSG begins with an acyclic graph, and any edges between global and strict transactions are added. Thus, any cycles created would involve at least one local transaction.

To break the cycle, global transactions require rollback, where a transaction $T$ can cause cascading aborts of local transactions that read the values that $T$ has written. In terms of graphs, this means that all transactions have a dependency edge from $T$. The rollback of a transaction is then required from the server that the transaction was propagated to, and the local SDB from which the transaction originated from. All other sites would not be affected since local transactions from those sites do not read the values that other transactions have seen.

The detection of cycles are difficult as polygraphs result when edges within a transaction that wrote an attribute value can have an arbitrary direction (Papadimitriou 1986). However, a polynomial test for acyclicity can be undertaken given the assumption that a transaction read an attribute value before writing it. Thus, only a precedence edge $ST \rightarrow GT$ is required to construct a GSG from a LSG, when a strict transaction $ST$ reads an item written by a global transaction $GT$. The steps for reconciliation is outlined in Algorithm 9.

**Algorithm 9** Reconciliation

```
while there are no cycles in GSG do
  rollback a global transaction to its origin, ie when it is a local transaction
  rollback all exact transactions related with a dependency edge to $GT$
end while
for all attribute values required by the transaction do
  propagate the current value to all lower tier SDB served by the server or
  CDB.
end for
```

Since it is possible for reconciliation to occur at a server SDB, if that server is consistent with the CDB the conflicts will be detected earlier and thus, limit the effects of cascading aborts.

## 8.4 Local Update Propagation

Since all attributes of a database are constrained to a certain size limit, updates to individual attribute values must result in values that are within the constraints. For these cases where local attribute values are updated with similar or smaller sized values a slight modification of the conventional ‘update everywhere’ approach as outlined by Gray et al. (1996) can be adopted. This is considered to be a Local Update Propagation (LUP) protocol. In order to perform the update, the client site is first required to request locks (see Section 8.2) on the pertinent attribute values, where given LUP, the transaction has the required attribute values and is therefore a local transaction. The local SDB is then able to grant locks for local transactions, and the client site can begin processing the update. Depending on the amount of storage left on the client site, the new values may be committed to the UDB. The results are subsequently propagated lazily to the central site. If committed to the UDB, the new values are immediately available to that client. Once committed on the central site, a database refresh (see Section 8.7) can be initiated by the central site, where the updated values are available on the client's SDB, either by moving it from the UDB or propagating it from the central site.

An insertion of new values into the local database can also be achieved through the LUP protocol, where the new data are created in the UDB and the results propagated to the central site. Once committed a database refresh is required to update both the SDB and its storage map to reflect the new information. For the deletion of existing attribute values from a local database, the LUP protocol is only used if the local database contains all the attribute values required of the transaction.

Of interest is the attribute-value sensitivity of the COSMOS database means that a change in value can change the interest in the value to a site. Thus a future refresh may result in that value not being retained in the SDB.

Once a transaction is locally committed, the transaction is propagated up the architecture to the CDB for processing, whereby each server is required to process the transaction and update the required attribute values locally. Depending on the locking mechanism used, the affected attribute values of all other descendent of this server would become *invisible* using the method described in the next section. This is to limit the effect of cascading aborts on those sites.

## 8.5 Update Through Local Item Invisibility

Given that the SDB is of limited size, a second scheme is necessary where updates would change the data such that the client's database becomes larger than the threshold originally placed upon it or where the local database does not contain enough data to process the update. When this occurs a relocation of the transaction is required, where it is converted into a global transaction. To ensure serialisability the lock is obtained before the transaction is relocated. These locks are granted from the local SDB if it is a local transaction, server SDB if it is a global transaction, and from the central site if it is a strict transaction. Since the correct application of the summarisation criteria will create a maximal SDB containing the information required by the user, it is assumed that relocation of transactions to a server would occur rarely.

A method for ensuring that a local value is not accessed, is to render the local copy of the data invisible by changing the storage map for the local SDB. That is, the data is effectively 'dropped' from the local database by masking out the relevant parts of the associated storage map. On receiving the update request, the client is required to convert the update into a SM, $U$, which is then considered an invisible mask for the SDB. Assuming the client's storage map of the SDB is $C$, then a simple bitwise operation of

$$C' = \neg(C \vee \neg U) \tag{8.1}$$

equates into an updated representation of the client database that does not include the items requested for update. Moreover,

$$U' = \neg(U \vee \neg C) \tag{8.2}$$

represents the bits of the query turned off. The client can also remove the physical data from its database, once those values have been committed into the central database. The steps that occur when a user places a request to update the database are:

1. User sends an update request to client.
2. Client converts the update into a storage map, $U$, making the relevant items invisible. $C'$ is used for all subsequent updates.
3. Client forwards the update request to the central database, and waits for a response.
4. If an update is committed, $C'$ is subsequently used. If update fails (or is rolled back) then $U'$ is used to reinstate old attribute values.
5. A partial push refresh is then initiated by the server to store the new values on to the UDB and the old data is systematically removed from the SDB.
6. When resources permit, a full database refresh operation is undertaken.

## 8.6 Bounded Divergence

For each site, the degree of divergence, $d$, for each attribute value represents the allowable difference between the same attribute values in a local SDB and its server, which is either the server SDB or CDB. These differences result from either the local SDB having local writes that were not propagated to its server, or updates that have not yet propagated from its server. As a result, it is possible to bound the degree of divergence by limiting the number of local writes until a global commit is performed. There are also different degrees of divergence between sites, that are of the same branch, but in different levels of the architecture. That is, a site closest to the CDB (one tier away) will have a smaller degree of divergence than another site of the same branch that is $n > 1$ tiers away. This is caused by the fact that if a server SDB is $k$-consistent with the CDB and a local SDB is $l$-consistent with the server SDB, then the local SDB is $k + l$-consistent with the CDB.

**Table 8.4. Protocols to maintain bounded inconsistencies**

| **Definition of divergence ($d$)** | **Possible protocols** |
|---|---|
| Maximum number of transactions that operate on local SDB copies. | Appropriately limit the number of local write transactions at the SDB. |
| Range of acceptable values that an attribute value may take. | Local transactions may only write to a certain range |
| Maximum number of attribute values that have divergent copies. | Appropriately limit the number of local attribute values that may be modified by local transactions. |

There are several options for defining the degree of divergence, $d$, each of which has a different implementation protocol, as shown in Table 8.4.

## 8.7 Database Refresh

Database refresh facilitates the ability to add information to a local SDB created through COSMOS, after the initial population. The CDB is required to hold all SMs associated with sites that are one tier away from it. Similarly, server SDB are required to hold SMs associated with sites directly below. The CDB and server SDB are then able to determine whether a given attribute value is of interest.

Three database refresh procedures are supported, partial pull, partial push and full refresh. The following sections discuss each in more detail.

**Partial Pull Refresh.**

This refresh is initiated where there are sufficient changes made to any criteria that the local site is currently using. Thus, it is a client-initiated operation. If a local site, such as a mobile host, is unable to store any information regarding its criteria, a connection with its server SDB is required before changes to the criteria and subsequent refresh is initiated. This will allow the server SDB to coordinate the calculation and refresh the local SDB.

When changes are made to the criteria for a local site, recalculation of the priorities of data affected by, the criteria involved and the criteria that it affects. Section 5.2 provides a discussion on the recalculation of the priority for each criteria. Once recalculated, the corresponding attribute values are included within the SDB, and the associated attribute values in the UDB are removed.

It is important to note that a strong connection between the local SDB and the CDB, through a server SDB, results in the local SDB being able to pull more data that it requires. When a local SDB has a connection to a server SDB, which does not have strong connectivity to the CDB, then the local SDB may only pull data available from the server. Any additional data is unavailable, and returned as a local null, until a connection can be established.

**Partial Push Refresh.**

This is a server-initiated refresh. When changes are committed on the CDB, the server will communicate the commit action with any affected sites and a database refresh is initiated. A recalculation of the priorities of those affected attribute values are required. The recalculation uses the same approach as partial pull refresh.

**Full Refresh.**

A full refresh involves a total recalculation of the priority table, which can be initiated from either a client or a server. This may occur when the client is expected to be in a long idle state, large bandwidth is available from the network, and/or the state of the local database has degraded so much that it is no longer useful. However, before a full refresh is initiated, any pending local transactions must be processed and globally committed. Once recalculated, the SDB should have all the required data it needs, and the entries in the UDB can be removed.

For all three refresh procedures, a new SM is created for the new local database after the priority table has been recalculated. The attribute values required can then be determined by calculating the difference between the new and current SMs, and placing a request to the server for additional data. The new SM is also sent to update the representation held at the server. Any additional attribute values not represented by the storage map are now permanently removed from the client's database.

## 8.8 System Recovery

To this point, the assumption was that the network connecting the clients to a server and the actual server, itself, does not fail. However, this is not always the case, especially when hardware errors occur or wireless networking is used. Since sites are hierarchical in COSMOS, where all lower tier sites are subsets of a higher tier site, it is possible for higher tier sites to serve even those that are more than one tier lower along its branch. Therefore, during a site disconnection, all the clients for that site can connect directly to its server. In the case where the central server is no longer accessible, recovery mechanisms are required.

The failure of the central site can be debilitating to a centralised distributed database system. At the least, it results in users being unable to initiate updates. Since the central site is used as a coordinator for any update request, a new coordinator must be chosen from the available sites to fulfil this role until the central site is functional. Selection of a new coordinator is achieved using the election algorithms proposed by Garcia-Molina (1982), in which sites nominates itself as the coordinator whenever it receives a specified number of failed communications with the current central coordinator. The site then attempts to communicate with all other operational sites informing them that it wants to be the new coordinator. When a majority of other sites agree, the nominated site becomes the coordinator.

In addition, the new coordinator will request the storage maps from all other directly connected sites. Since the allocation of data is such that each site only contains information useful to them, the election algorithm must be extended to include the nomination of coordinators to service different sections of the data. This is achieved by enabling the new coordinator to assign additional coordinators. Therefore, when a transaction request is received that includes data that the new coordinator does not have, it attempts to search all replica sites, by their SMs, for sites that contain a majority, if not all, of the attribute values required by the transaction. Once found, the site becomes a secondary coordinator and a

SM is generated to represent the attribute values that it is now responsible for. This site is then assigned a priority by the primary coordinator based upon the amount of data for which it is responsible for. The SM generated on a secondary site is distinct from the primary coordinator's or any other secondary sites. A copy of which is kept with the primary coordinator ensuring that it knows where to direct data requests. The secondary coordinator will subsequently attempt to request for additional attribute values that it does not have to complete the transaction from the primary. Multiple secondary coordinator is possible depending on the number of sites directly connected to the primary.

In the event that the new coordinator fails before the central site is operational, a secondary site with the highest priority takes over as primary. Assuming that consistency is guaranteed between all running sites, the failure of a secondary site will have little impact, since the primary may assign a new secondary site to take its place. The worst-case scenario occurs when data is unavailable at any of the sites still running to process a transaction. In this case, the transaction is postponed until the central site becomes operational.

A network partition is used to describe a network split, such that sites can only communicate with sites within the same partition. When this occurs, sites within the CDB partition will work as normal while sites in other partitions will presume that a failure has occurred upon the central site. This will result in the election of a replacement for the CDB. This allows operations to continue on that partition, and ensures consistency between sites within each partition. Since network partitioning is not limited to two partitions, each partition must assign its own primary site.

In either case, where the central site fails or a network partition occurs and the central site is perceived to have failed, a synchronisation process must occur between the central site and any existing primary and secondary sites when network communications are possible between them. However, if two or more partitioned networks are able to communicate with each other before the central site is operational then merging occurs, in which the network with highest priority primary, takes precedence over the other primary candidates. The primary site of the highest priority is now the new primary site, and all previous updates are now propagated and synchronised with the new primary site. In the event that two or more primary sites have the same priority, then the total priority of the primary and all secondary sites is calculated for each network and the highest will take precedence.

## 8.9 Summary

Protocols are rules ensuring that a system operates the way it is supposed to. This chapter discussed the protocols for transaction management, update propagation and system recovery. For transaction management, protocols are required to ensure that the transactions operations are mapped to the correct database, to ensure serialisability of schedules and to ensure correct execution of reconciliation. Additionally, protocols for keeping the inconsistency between databases at different sites bounded are also presented.

The use of protocols, especially those used in the management of transactions, affect transaction response times. Thus, the next chapter (Chapter 9) will discuss a quantitative evaluation of COSMOS.

# Chapter 9

# Evaluation of COSMOS

The evaluation of what is stored within a SDB is usually subjective and subsequently it is quite difficult to measure usefulness a SDB since the needs of each user are different. Thus, a quantitative evaluation of COSMOS is presented in this chapter, focusing on an evaluation of the transaction processes involved.

This chapter is structured as follows: Section 9.1 examines quantitative evaluation methods, in particular the choices required when designing a performance model. Section 9.2 discusses the model used in this chapter. Section 9.3 provides the results of an analytical analysis of COSMOS. Section 9.4 provide a simulation of COSMOS. Finally, Section 9.5 provides a discussion of this chapter.

## 9.1 Quantitative Evaluations

Nicola and Jarke (2000) presented a comprehensive survey of performance modelling of distributed and replicated databases, where several basic choices for quantitative evaluation models are proposed. The first relates to the type of performance criteria required. The most common of which considered are the transaction response time and the transaction throughput. Other performance metrics used to measure distributed systems include number of messages, hardware cost, availability and transaction abort rates (Alonso et al. 1990, Barbara & Garcia-Molina 1982, Cheung, Ammar & Ahamad 1992, Ciciani, Dias & Yu 1990, Pitoura & Bhargava 1995*b*).

The second is a choice between an analytical model, a simulation or a hybrid of both. The main advantage of simulations is that they are able to evaluate system models are too complex for analytical models. However, simulations are costly in terms of programming and computing time. Analytical models, on the other hand, are capable of obtaining performance results efficiently from closed form expressions and numerical iterations.

The third choice is parameter values. Every performance model requires a set of input parameters. These parameters may be dependent on currently available technologies such as disk service time and CPU speed, or dependent on the application, such as the number database nodes and number of operations per transaction. The former is usually easier to accurately estimate than the latter.

The final choice is the homogeneity assumption, which is common to nearly all types of performance models. That is, the homogeneity assumption is when all database sites and their respective workloads are identical. The inter-database activities are also symmetrical among participating sites. This assumption implies that all databases have the same structure and service capacity, that all sites have the same transaction arrival rate, or the communication between sites are symmetrical.

## 9.2 Performance Model

It is assumed that a COSMOS distributed system will have $n$ major sites and a Poisson arrival rate for both queries and updates. That is, $\lambda_q$ and $\lambda_u$ are the average rate of queries and updates on attribute values initiated at each site.

For the proposed model, the central site contains the main database copy and receives and processes all strict transactions. Let $a$ be the number of tiers a site is away from the central site, where $a = 0$ specifies the central site, and $A$ is the total number of tiers in the system architecture. Assume that an average fraction, $h$, of the database is summarised onto the tier-1 sites, while tier-2 sites have an average summarisation fraction of $k_2h$. Thus, tier-$a$ has an average summarisation fraction of $k_ah$ where $k_1 = 1$. In a worst case scenario, the smallest database would be a local SDB at the tier-$A$ and would have an average fraction of $k_Ah$. Thus, the smallest server SDB has an average fraction of $k_{A-1}h$. For the purpose of this evaluation, all references to the local and server SDB imply the furthest tiered local and server SDB, respectively.

Let $c$ be the consistency value such that it represents the fraction of the arrived operations to the system are strict. That is, for $c = 1$ all arrived operations are strict.

If the attribute values required by a transaction are random, then the transaction will be processed locally with a probability of $ah$. However, since transactions are more likely to access summarised data, let $f$ denote the probability that the transaction will access database *hot spots. Hot spots* in this case is what is available locally at the SDB. Hence, a transaction has a $k_a hf$ probability of being locally processed and a $1 - k_a hf$ chance of being propagated to the central site.

### 9.2.1 Resource Contention Analysis

Sites are modelled using M/G/1 servers, where each request is processed with the same priority on a first-come, first-served basis. The number of sites, $n$ is determined by

$$n = \sum_{i=0}^{A} [d^i] \tag{9.1}$$

where $d$ is the average number of sites.

The average service time for the various types of request, exponentially distributed, is determined using the following parameters:

- $t_q^C$ processing time of a query on a summary copy at the CDB.
- $t_u^C$ processing time of installing an update on a summary copy at the CDB.
- $t_q^S$ processing time of a query on a summary copy at a server SDB.
- $t_u^S$ processing time of installing an update on a summary copy at a server SDB.
- $t_q^L$ processing time of a query on a summary copy at a local SDB.
- $t_u^L$ processing time of installing an update on a summary copy at a local SDB.
- $t_b$ overhead time to propagate an update or query to another server SDB.

Since each site's connectivity is unreliable, weak connections must also be modelled. Pitoura and Bhargava (1999) proposed the modelling of disconnections by using a M/M/1 server with vacations whereby a vacation system is one which the server accordingly becomes unavailable for a period of time. For such system, given that messages have an average size of $m$ with exponentially distributed packet lengths and an available bandwidth of $W$, the service rate $s_r$ is $W/m$.

Now, the total number of messages transmitted per second, $M$, between sites are:

$$M = 2n[c(\lambda_q + \lambda_u) + (1-c)(1-k_A hf)(\lambda_q + \lambda_u)] \tag{9.2}$$

where the first term corresponds to the propagation of strict traffic to the CDB, and the second term corresponds to the propagation of global query traffic up one tier higher.

To calculate execution time, the communication overheads between strongly connected sites are ignored and it is assumed that grouped together they contain a single server node with relatively fast connections.

For each site, there are the following types of request:

1. Locally initiated queries, $\lambda_1 = (1-c)\lambda_q$ are local and are serviced locally with an average service time of $\theta_1 = t_q^l$.
2. Local updates arrive at a rate of $\lambda_2 = (1-c)k_A hf\lambda_u$ and have an average service time of $\theta_2 = t_u^l$.
3. Strict updates arrive at a rate of $\lambda_3 = k_A hfc\lambda_u$ with an average service time of $\theta_3 = t_u^l$.

$$E[X^l] = \sum_{i=1}^{3}(\frac{\lambda_i}{\lambda})\theta_i \tag{9.3}$$

$$E[(X^l)^2] = \sum_{i=1}^{3}(\frac{\lambda_i}{\lambda})2\theta_i^2 \tag{9.4}$$

Similarly, for server sites, the wait time is calculated using the following types of requests:

- Local queries arrive at a rate of $\lambda_1 = (1-c)k_{A-1}hf\lambda_q$ with an average service time of $\theta_1 = t_q^s$.
- Global queries arrive at a rate of $\lambda_2 = (1-c)[1-k_A hf]\lambda_q$ with an average service time of $\theta_2 = t_q^s$.
- Local updates are installed at a rate of $\lambda_3 = (1-c)k_{A-1}hf\lambda_u$ and an average service time of $\theta_3 = t_u^s$.
- Global updates arrive at a rate of $\lambda_4 = (1-c)[1-k_A hf]\lambda_u$ with an average service time of $\theta_4 = t_u^s$.

- Strict updates are installed at a rate of $\lambda_5 = k_{A-1}hfc\lambda_u$ with an average service time of $\theta_5 = t_u^s$.
- Strict updates are propagated to servers and clients one tier away at a rate of $\lambda_6 = k_A hfc\lambda_u$ with an average service time of $\theta_6 = dt_b$.

The service time of the combined flow, $X^s$, is no longer exponentially distributed but has a mean and second moment of:

$$E[X^s] = \sum_{i=1}^{6}(\frac{\lambda_i}{\lambda})\theta_i \tag{9.5}$$

$$E[(X^s)^2] = \sum_{i=1}^{6}(\frac{\lambda_i}{\lambda})2\theta_i^2 \tag{9.6}$$

Finally, for the central site, the wait time may be calculated using the following types of requests:

- Strict queries arrive at a rate of $\lambda_1 = c\lambda_q$ with an average service time of $\theta_1 = t_q^c$.
- Strict updates are installed at a rate of $\lambda_2 = c\lambda_u$ and an average service time of $\theta_2 = t_u^c$ .
- Strict updates are propagated to servers and clients one tier away at a rate of $\lambda_3 = hfc\lambda_u$ with an average service time of $\theta_3 = dt_b$.

The service time of the combined flow, $X^c$, is no longer exponentially distributed but has a mean and second moment of:

$$E[X^c] = \sum_{i=1}^{3}(\frac{\lambda_i}{\lambda})\theta_i \tag{9.7}$$

$$E[(X^c)^2] = \sum_{i=1}^{3}(\frac{\lambda_i}{\lambda})2\theta_i^2 \tag{9.8}$$

Thus, using the Pollaczek-Khinchin formula, the wait time, $w$ is:

$$w = \frac{\lambda E[(X^c)^2]}{2(1 - \lambda E[X^c])} \tag{9.9}$$

The wait times for the different types of site are therefore:

$$\text{local sites:} \quad w_L = \tfrac{\sum_{i=1}^{3} \lambda_i \theta_i^2}{1-\sum_{i=1}^{3} \lambda_i \theta_i} \tag{9.10}$$

$$\text{server sites:} \quad w_S = \tfrac{\sum_{i=1}^{6} \lambda_i \theta_i^2}{1-\sum_{i=1}^{6} \lambda_i \theta_i} \tag{9.11}$$

$$\text{central site:} \quad w_C = \tfrac{\sum_{i=1}^{3} \lambda_i \theta_i^2}{1-\sum_{i=1}^{3} \lambda_i \theta_i} \tag{9.12}$$

The average response time, without considering data contention, for a local read ($R_q^L$), local write ($R_w^L$), global read ($R_q^G$), global write ($R_w^G$), strict read ($R_q^S$) and strict write ($R_w^S$) are as follow:

$$R_q^L : \quad R \overset{L}{q} = (w_L + t_q^l) k_A h f \tag{9.13}$$

$$R_w^L : \quad R \overset{L}{u} = (w_L + t_u^l) k_A h f \tag{9.14}$$

$$R_q^G : \quad R \overset{G}{q} = (w_S + t_q^s + t_r)(1 - k_{A-1} h f) \tag{9.15}$$

$$R_w^G : \quad R \overset{G}{u} = (w_S + t_u^s + t_r)(1 - k_{A-1} h f) \tag{9.16}$$

$$R_q^S : \quad R \overset{S}{q} = w_C + t_q^c + d t_r \tag{9.17}$$

$$R_w^S : \quad R \overset{S}{u} = w_C + t_u^c + d t_r \tag{9.18}$$

Each network link is modelled similarly to Pitoura and Bhargava's (1999), where the arrival rate, $\lambda_r$, for each link is Poisson with a mean of $M/(n(n-1))$. The average transmission waiting time $w_r$, of a non-exhaustive vacation system, or a queue system with Bernoulli scheduling (Takagi 1991), is used. That is, after the end of each service, the server continues its service with a probability of $p$, or takes a vacation with a probability of $1-p$. This is modelled by:

$$w_r = \frac{E[v^2]}{2E[v]} + \frac{\lambda_r \{ s_r^{(2)} + (1-p)(2(1/s_r)E[v] + E[v^2]) \}}{2\{1 - p - (1-p)\lambda_r E[v]\}} \tag{9.19}$$

where $s_r^{(2)}$ is the second moment of the service rate and $v$ is the duration of disconnection. The average transmission time $t_r$ is then the sum of the service time, $1/s_r$ and the wait time, $w_r$, at each network link, obtained by:

$$t_r = 1/s_r + w_r \tag{9.20}$$

### 9.2.2 Data Contention Analysis

For data contention analysis, a transaction is modelled by having $n_L + 2$ states (Yu, Dias & Lavenberg 1993), where $n_L$ is the random number of attribute values accessed by the transaction. State 0 refers to the initial setup, while the execution phases, states $1, 2, ..., n_L$, are executed in that order. State $n_L + 1$ terminates the process with the transaction entering a commit phase if successful. The transaction response time $r_{trans}$ is expressed as (Yu et al. 1993):

$$r_{trans} = r_{INPL} + r_E + \sum_{j=1}^{n_W} r_{w_j} + t_{commit} \tag{9.21}$$

where

- The execution time in state 0 is $r_{INPL}$.
- The sum of the execution time in states $1, 2, .., n_L$ excluding lock waiting times is $r_E$.
- The number of lock waits during the run of a transaction is $n_W$.
- The wait time for the $j$th lock contention is $r_{w_j}$.
- The commit time to reflect the updates in the database is $t_{commit}$.

Then, using Formula 9.21, the response time for strict, global and local transactions is [1]:

$$R_{strict-transaction} = R_{INPL} + R_{E_{strict}} + N_q P_{SR} R_{SR} + N_u P_{SW} R_{SW} + T_{commit} \tag{9.22}$$

$$R_{global-transaction} = R_{INPL} + R_{E_{global}} + N_q P_{GR} R_{GR} + N_u P_{GW} R_{GW} + T_{commit} \tag{9.23}$$

$$R_{local-transaction} = R_{INPL} + R_{E_{local}} + N_q P_{LR} R_{LR} + N_u P_{LW} R_{LW} + T_{commit} \tag{9.24}$$

where $P_{op}$ is the probability that a transaction contends for an *op* operation on a data copy. $R_{op}$ is the average waiting time to get lock for *op*.

[1] where lowercase letters represent random variables and uppercase letters represent the average of the corresponding random variables.

The term $R_{E_{op}}$ can be approximated using resource contention analysis, by:

$$R_{E_{strict}} = N_q R_q^S + N_u R_u^S \tag{9.25}$$

$$R_{E_{global}} = N_q R_q^G + N_u R_u^G \tag{9.26}$$

$$R_{E_{local}} = N_q R_q^L + N_u R_u^L \tag{9.27}$$

The approximation of $P_{op}$ and $R_{op}$ considers the locks held by a transaction. Each state $i$ of a local transaction is divided into subtransactions where $i_1$ is the lock state and $i_2$ is the execution state. In substate $i_0$ the transaction is at its initiating site, holds $i-1$ locks and sends messages to the server SDB or CDB. In substate $i_1$, the transaction holds $i-1$ locks and is waiting for the $i$th set of locks. In substate $i_2$ the transaction now holds $i$ locks and is executing.

The probability that substate $i_1$ is entered after leaving substate $i_2 - 1$ or $i_0$ is the lock contention probability, $P_{LR}$, $P_{LW}$, $P_{GR}$, $P_{GW}$, $P_{SR}$, $P_{SW}$ for LR, LW, GR, GW, SR and SW lock requests, respectively. Let $c_{op}$ be the average time an *op* spends in state $i_0$. Thus, for $GR$, $c_{GR} = w_S + t_b + t_r$. Similarly, $c_{GW} = w_S + t_b + t_r$, $c_{SR} = w_C + t_b + at_r$, and $c_{SW} = w_C + t_b + at_r$. Let $a_{op}$ be the average time in substate $i_2$, which is determined by using the average response time from the resource contention analysis. Thus, $a_{LR} = R_q^L = (w_L + t_q^l)k_a hf$, $a_{LW} = R_u^L$, $a_{GR} = R_q^G$, $a_{GW} = R_u^G$, $a_{SR} = R_q^S$, and $a_{SW} = R_u^S$. Finally, the time spent in substate $i_1$ is $R_{op}$, and the unconditional mean time in substate $i_1$ is thus $b_{op}$, such that $b_{LR} = P_{LR}R_{LR}$, $b_{LW} = P_{LW}R_{LW}$, $b_{GR} = P_{GR}R_{GR}$, $b_{GW} = P_{GW}R_{GW}$, $b_{SR} = P_{SR}R_{SR}$, and $b_{SW} = P_{SW}R_{SW}$.

Let $P_{op_1/op_2}$ be the probability that an $op_1$-lock request conflicts with an $op_2$-lock request, and $T_L$ ($T_G$ and $T_S$) be the average lock holding time for local (global and strict) transactions. Then,

$$\begin{aligned} P_{LR} &= P_{LR/LW} \\ &= k_A hf \lambda_q (1-c) T_L \end{aligned} \tag{9.28}$$

Similarly,

$$\begin{aligned} P_{LW} &= P_{LW/LR} + P_{LW/LR} \\ &= k_A hf(1-c)(\lambda_q + \lambda_u) T_L \end{aligned} \tag{9.29}$$

$$P_{GR} = k_{A-1} hf(1-c)\lambda_q T_G \tag{9.30}$$

$$P_{GW} = k_{A-1}hf(1-c)(\lambda_q + \lambda_u)T_G \tag{9.31}$$

$$P_{SR} = (1-hf)c\lambda_q T_S \tag{9.32}$$

$$P_{SW} = (1-hf)c(\lambda_q + \lambda_u)T_S \tag{9.33}$$

Let $G_L$, $G_G$ and $G_S$ be the the sum of the average lock holding times over all N copies accessed by a local, global and strict transactions, respectively. Then,

$$G_L = \{\sum_{i=1}^{N}[\frac{N_q}{N}(ia_{LR} + (i-1)b_{LR}) + \frac{N_u}{N}(ia_{LW} + (i-1)b_{LW})]\} + NT_{c_L} \tag{9.34}$$

$$\begin{aligned} G_G &= \{\sum_{i=1}^{N}[\frac{N_q}{N}(ia_{GR} + (i-1)b_{GR} + (i-1)c_{GR}) \\ &+ \frac{N_u}{N}(ia_{GW} + (i-1)b_{GR} + (i-1)c_{GW})]\} + NT_c \end{aligned} \tag{9.35}$$

$$\begin{aligned} G_S &= \{\sum_{i=1}^{N}[\frac{N_q}{N}(ia_{SR} + (i-1)b_{SR} + (i-1)c_{SR}) \\ &+ \frac{N_u}{N}(ia_{SW} + (i-1)b_{SW}) + (i-1)c_{SW}]\} + NT_c \end{aligned} \tag{9.36}$$

where $T_c$ is the average commit time. Then, $T_L = G_L/N$, $T_G = G_G/N$ and $T_S = G_S/N$ for local, global and strict transactions, respectively. Let $CP^{i_e}_{op1/op2}$ be the conditional probability that an *op*1-lock request contents with a transaction in substate $i_e$, which holds an incompatible *op*2-lock, given that lock contention does occur. Let *optype*1 be the read or write operation of *op*1 and *opt*1*cat* be the operation's category. For example, if *op*1 is a LR then *optype*1 has a 'q' (query value), and *opt*1*cat* has the value 'local', then,

$$CP^{i_0}_{op1/op2} = (i-1)[(N_{optype1}/N * c_{op1} + (N_{optype2}/N) * c_{op2}]/G_{op1cat} \tag{9.37}$$

Let the factors $f_i$ then express the average remaining times at each corresponding substate, where its value is dependant on the distribution of the substate times. Additionally, if $s_{L_i}$, $s_{G_i}$ and $s_{S_i}$ are the average time for a local, global and strict transaction, respectively, from acquiring the $i$th lock until the end of commit, then

$$s_{L_i} = (N-i)[\frac{N_q}{N}(a_L R + b_L R) + \frac{N_u}{N}(a_L W + b_L W)] + T_c \quad (9.38)$$

$$s_{G_i} = (N-i)[\frac{N_q}{N}(a_G R + b_G R + c_G R) + \frac{N_u}{N}(a_G W + b_G W + C_G W)] + T_c \quad (9.39)$$

$$s_{S_i} = (N-i)[\frac{N_q}{N}(a_S R + b_S R + c_S R) + \frac{N_u}{N}(a_S W + b_S W + C_S W)] + T_c \quad (9.40)$$

The average time spent waiting to get an *op* lock is then $R_{op}$ (given the locking matrix in Table 8.2) and they are as follows:

$$\begin{aligned} R_{LR} &= \{\sum_{i=1}^{N}[CP^{i_1}_{LR/LW}(\frac{R_{LW}}{f_1} + a_{LW} + s_{L_i}) + CP^{i_2}_{LR/LW}(\frac{a_{LW}}{f_2} + s_{L_i}) \\ &+ \frac{NT_{c_L}}{G_L}(\frac{T_c}{f_3}) \end{aligned} \quad (9.41)$$

$$\begin{aligned} R_{LW} &= \{\sum_{i=1}^{N}[CP^{i_1}_{LW/LW}(\frac{R_{LW}}{f_1} + a_{LW} + s_{L_i}) + CP^{i_2}_{LW/LW}(\frac{a_{LW}}{f_2} + s_{L_i}) \\ &+ CP^{i_1}_{LW/LR}(\frac{R_{LR}}{f_1} + a_{LR} + s_{L_i}) + CP^{i_2}_{LW/LR}(\frac{a_{LR}}{f_2} + s_{L_i}) \\ &+ \frac{NT_c}{G_L}(\frac{T_c}{f_3}) \end{aligned} \quad (9.42)$$

$$\begin{aligned} R_{GR} &= \{\sum_{i=1}^{N}[CP^{i_0}_{GR/GW}(\frac{c_{GW}}{f_0} + R_{GW} + a_{GW} + s_{G_i}) + CP^{i_1}_{GR/GW}(\frac{R_{GW}}{f_1} + a_{GW} + s_{G_i}) \\ &+ CP^{i_2}_{GR/GW}(\frac{a_{GW}}{f_2} + s_{G_i})]\} \\ &+ \frac{NT_c}{G_G}(\frac{T_c}{f_3}) \end{aligned} \quad (9.43)$$

$$\begin{aligned}
R_{GW} &= \{\sum_{i=1}^{N}[CP^{i_0}_{GW/GW}(\frac{c_{GW}}{f_0} + R_{GW} + a_{GW} + s_{G_i}) + CP^{i_1}_{GW/GW}(\frac{R_{GW}}{f_1} + a_{GW} + s_{G_i}) \\
&+ CP^{i_2}_{GW/GW}(\frac{a_{GW}}{f_2} + s_{G_i}) + CP^{i_0}_{GW/GR}(\frac{c_{GR}}{f_0} + R_{GR} + a_{GR} + s_{G_i}) \\
&+ CP^{i_1}_{GW/GR}(\frac{R_{GR}}{f_1} + a_{GR} + s_{G_i}) + CP^{i_2}_{GW/GR}(\frac{a_{GR}}{f_2} + s_{G_i})]\} \\
&+ \frac{NT_c}{G_G}(\frac{T_c}{f_3})
\end{aligned} \tag{9.44}$$

$$\begin{aligned}
R_{SR} &= \{\sum_{i=1}^{N}[CP^{i_0}_{SR/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) \\
&+ CP^{i_1}_{SR/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) + CP^{i_2}_{SR/SW}(\frac{a_{SW}}{f_2} + s_{S_i})]\} \\
&+ \frac{NT_c}{G_S}(\frac{T_c}{f_3})
\end{aligned} \tag{9.45}$$

$$\begin{aligned}
R_{SW} &= \{\sum_{i=1}^{N}[CP^{i_0}_{SW/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) + CP^{i_1}_{SW/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) \\
&+ CP^{i_2}_{SW/SW}(\frac{a_{SW}}{f_2} + s_{S_i}) + CP^{i_0}_{SW/SR}(\frac{c_{SR}}{f_0} + R_{SR} + a_{SR} + s_{S_i}) \\
&+ CP^{i_1}_{SW/SR}(\frac{R_{SR}}{f_1} + a_{SR} + s_{S_i}) + CP^{i_2}_{SW/SR}(\frac{a_{SR}}{f_2} + s_{S_i})]\} \\
&+ \frac{NT_c}{G_S}(\frac{T_c}{f_3})
\end{aligned} \tag{9.46}$$

For the locking matrix in Table 8.3, $R_{op}$ are determined as follows:

$$\begin{aligned}
R_{LR} &= \{\sum_{i=1}^{N}[CP^{i_1}_{LR/LW}(\frac{R_{LW}}{f_1} + a_{LW} + s_{L_i}) + CP^{i_2}_{LR/LW}(\frac{a_{LW}}{f_2} + s_{L_i}) \\
&+ CP^{i_0}_{LR/GW}(\frac{c_{GW}}{f_0} + R_{GW} + a_{GW} + s_{G_i}) + CP^{i_1}_{LR/GW}(\frac{R_{GW}}{f_1} + a_{GW} + s_{G_i}) \\
&+ CP^{i_2}_{LR/GW}(\frac{a_{GW}}{f_2} + s_{G_i}) + CP^{i_0}_{LR/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) \\
&+ CP^{i_1}_{LR/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) + CP^{i_2}_{LR/SW}(\frac{a_{SW}}{f_2} + s_{S_i})]\} \\
&+ \frac{NT_c}{G_L}(\frac{T_c}{f_3})
\end{aligned} \tag{9.47}$$

$$\begin{aligned}
R_{LW} &= \{\sum_{i=1}^{N}[CP^{i_1}_{LW/LR}(\frac{R_{LR}}{f_1} + a_{LR} + s_{L_i}) + CP^{i_2}_{LW/LR}(\frac{a_{LR}}{f_2} + s_{L_i}) \\
&+ CP^{i_1}_{LW/LW}(\frac{R_{LW}}{f_1} + a_{LW} + s_{L_i}) + CP^{i_2}_{LW/LW}(\frac{a_{LW}}{f_2} + s_{L_i}) \\
&+ CP^{i_0}_{LW/GW}(\frac{c_{GW}}{f_0} + R_{GW} + a_{GW} + s_{G_i}) + CP^{i_1}_{LW/GW}(\frac{R_{GW}}{f_1} + a_{GW} + s_{G_i}) \\
&+ CP^{i_2}_{LW/GW}(\frac{a_{GW}}{f_2} + s_{G_i}) + CP^{i_0}_{LW/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) \\
&+ CP^{i_1}_{LW/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) + CP^{i_2}_{LW/SW}(\frac{a_{SW}}{f_2} + s_{S_i})]\} \\
&+ \frac{NT_c}{G_L}(\frac{c}{f_3})
\end{aligned} \tag{9.48}$$

$$\begin{aligned}
R_{GR} &= \{\sum_{i=1}^{N}[CP^{i_0}_{GR/GW}(\frac{c_{GW}}{f_0} + R_{GW} + a_{GW} + s_{G_i}) + CP^{i_1}_{GR/GW}(\frac{R_{GW}}{f_1} + a_{GW} + s_{G_i}) \\
&+ CP^{i_2}_{GR/GW}(\frac{a_{GW}}{f_2} + s_{G_i}) + CP^{i_0}_{GR/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) \\
&+ CP^{i_1}_{GR/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) + CP^{i_2}_{GR/SW}(\frac{a_{SW}}{f_2} + s_{S_i})]\} \\
&+ \frac{NT_c}{G_G}(\frac{c}{f_3})
\end{aligned} \tag{9.49}$$

$$\begin{aligned}
R_{GW} &= \{\sum_{i=1}^{N}[CP^{i_1}_{GW/LR}(\frac{R_{LR}}{f_1} + a_{LR} + s_{G_i}) + CP^{i_2}_{GW/LR}(\frac{a_{LR}}{f_2} + s_{G_i}) \\
&+ CP^{i_1}_{GW/LW}(\frac{R_{LW}}{f_1} + a_{LW} + s_{G_i}) + CP^{i_2}_{GW/LW}(\frac{a_{LW}}{f_2} + s_{G_i}) \\
&+ CP^{i_0}_{GW/GR}(\frac{c_{GR}}{f_0} + R_{GR} + a_{GR} + s_{G_i}) + CP^{i_1}_{GW/GR}(\frac{R_{GR}}{f_1} + a_{GW} + s_{G_i}) \\
&+ CP^{i_2}_{GW/GR}(\frac{a_{GR}}{f_2} + s_{G_i})CP^{i_0}_{GW/GW}(\frac{c_{GW}}{f_0} + R_{GW} + a_{GW} + s_{G_i}) \\
&+ CP^{i_1}_{GW/GW}(\frac{R_{GW}}{f_1} + a_{GW} + s_{G_i}) + CP^{i_2}_{GW/GW}(\frac{a_{GW}}{f_2} + s_{G_i}) \\
&+ CP^{i_0}_{GW/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) + CP^{i_1}_{GW/SW}(\frac{R_{SW}}{f_1} \\
&+ a_{SW} + s_{S_i}) + CP^{i_2}_{GW/SW}(\frac{a_{SW}}{f_2} + s_{S_i})]\} + \frac{NT_c}{G_G}(\frac{c}{f_3})
\end{aligned} \tag{9.50}$$

$$\begin{aligned}
R_{SR} &= \{\sum_{i=1}^{N}[CP^{i_0}_{SR/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) \\
&+ CP^{i_1}_{SR/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) + CP^{i_2}_{SR/SW}(\frac{a_{SW}}{f_2} + s_{S_i})]\} \\
&+ \frac{NT_c}{G_S}(\frac{T_c}{f_3})
\end{aligned} \tag{9.51}$$

$$\begin{aligned}
R_{SW} &= \{\sum_{i=1}^{N}[CP^{i_1}_{SW/LR}(\frac{R_{LR}}{f_1} + a_{LR} + s_{S_i}) + CP^{i_2}_{SW/LR}(\frac{a_{LR}}{f_2} + s_{S_i}) \\
&+ CP^{i_1}_{SW/LW}(\frac{R_{LW}}{f_1} + a_{LW} + s_{S_i}) + CP^{i_2}_{SW/LW}(\frac{a_{LW}}{f_2} + s_{S_i}) \\
&+ CP^{i_0}_{SW/GR}(\frac{c_{GR}}{f_0} + R_{GR} + a_{GR} + s_{S_i}) + CP^{i_1}_{SW/GR}(\frac{R_{GR}}{f_1} + a_{GR} + s_{S_i}) \\
&+ CP^{i_2}_{SW/GR}(\frac{a_{GR}}{f_2} + s_{S_i}) + CP^{i_0}_{SW/GW}(\frac{c_{GW}}{f_0} + R_{GW} + a_{GW} + s_{S_i}) \\
&+ CP^{i_1}_{SW/GW}(\frac{R_{GW}}{f_1} + a_{GW} + s_{S_i}) + CP^{i_2}_{SW/GW}(\frac{a_{GW}}{f_2} + s_{S_i}) \\
&+ CP^{i_0}_{SW/SR}(\frac{c_{SR}}{f_0} + R_{SR} + a_{SR} + s_{S_i}) + CP^{i_1}_{SW/SR}(\frac{R_{SR}}{f_1} + a_{SR} + s_{S_i}) \\
&+ CP^{i_2}_{SW/SR}(\frac{a_{SR}}{f_2} + s_{S_i}) + CP^{i_0}_{SW/SW}(\frac{c_{SW}}{f_0} + R_{SW} + a_{SW} + s_{S_i}) \\
&+ CP^{i_1}_{SW/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) + CP^{i_2}_{SW/SW}(\frac{a_{SW}}{f_2} + s_{S_i})]\} \\
&+ \frac{NT_c}{G_S}(\frac{c}{f_3})
\end{aligned} \tag{9.52}$$

## 9.2.3 Reconciliation

The cost of restoring consistency is estimated in terms of the number of local transactions required to be rolled back, given the size of the CDB. The method described in this section extends Pitoura and Bhargava (1999).

For the CDB site, a global transaction, $GT$, is rolled back if its writes conflict with a read operation of a strict transaction, $ST$, that follows it in a GSG. Let $P_1$ be the probability that a $GT$ writes to an attribute value read by a $ST$, and $P_2$ be the probability that a $ST$ follows a $WT$ in the serialisation graph. Then, the probability that a global transaction is rolled back is $P = P_1P_2$. Given that reconciliation occurs after $N_r$ transactions, of which $y = cN_r$ are strict and $y' = (1-c)N_r$ are global. Assuming a uniform access distribution for simplicity (Yu et al. 1993). Then,

$$P_1 \simeq 1 - (1 - N_u/hD)^{N_q} \tag{9.53}$$

where $D$ is the average number of attribute values at a site one tier away from the CDB.

Let $p_{KL}$ be the probability that, in the GSG, there exists an edge from a given transaction of category $K$ to transaction of category $L$, where a category is either local, global or strict. Let $p'_{KL}(m, m')$ then be the probability that in a GSG with $m$ strict and $m'$ global transactions, there is an edge from a given transaction of category $K$ to any transaction of category $L$. The following probability of edges then exist:

$$\begin{aligned} p_{GG} &= [1-(1-\frac{N_u}{D})^N]p_c && (9.54) \\ p'_{GG}(m,m') &= 1-(1-p_{GG})^{(m'-1)} && (9.55) \end{aligned}$$

where $p_c = 1/n^2$ is the probability that two given transactions are initiated from the same branch.

$$\begin{aligned} p_{GS} &= 1-(1-\frac{N_q}{nhD})^{N_u h} && (9.56) \\ p'_{GS}(m,m') &= 1-(1-p_{GS})^{m} && (9.57) \end{aligned}$$

$$\begin{aligned} p_{SG} &= p_{GS} && (9.58) \\ p'_{SG}(m,m') &= 1-(1-p_{SG})^{m'} && (9.59) \end{aligned}$$

$$\begin{aligned} p_{SS} &= 1-(1-\frac{N_u}{D})^{N} && (9.60) \\ p'_{SS}(m,m') &= 1-(1-p_{SS})^{(m-1)} && (9.61) \end{aligned}$$

Let p(m,m',i) then be the probability that there is an acyclic path of length $i$. That is, a path with $i+1$ distinct nodes, that begins at a global transaction and terminates at a strict transaction in a GSG with $m$ strict and $m'$ global transactions. Then,

$$P_2 = 1 - \prod_{i=0}^{y+y'-1} [1 - p(y, y', i)] \qquad (9.62)$$

An iterative method is used to calculate the value of $p(y, y', i)$ using the following recursive relations (Pitoura & Bhargava 1999):

$$\begin{aligned} p(m,m',1) &= p_{GS} \text{ for all } m > 0, m' > 0 && (9.63) \\ p(m,m',0) &= 0 \text{ for all } m > 0, m' > 0 && (9.64) \\ p(m,0,i) &= 0 \text{ for all } m > 0, i > 0 && (9.65) \\ p(0,m',i) &= 0 \text{ for all } m' > 0, i > 0 && (9.66) \end{aligned}$$

$$p(y, y', i+1) = 1 - [(1 - p'_{GG}(k, k')p(k, k'-1, i)) \\ (\prod_{j=1}^{i-1} 1 - (p'_{GS}(k, k') \prod_{l=1}^{i-j-1} p'_{SS}(k-l, k'-1) \\ p'_{SG}(k-i+j, k'-1) \quad p(k-i+j-1, k'-2, j)) \\ (1 - p'_{WS}(k, k') \prod_{l=1}^{i-1} p'_{SS}(k-l, k'-1)p_S S)] \tag{9.67}$$

The first term represents the probability of a path whose first edge is between global transactions. The second term is the probability of a path whose first edge is between a global and strict transaction and includes at least one more global transaction than strict transaction. The last term is the probability of a path whose first edge is between a global and a strict transaction and does not include any other global transactions. Thus, the actual number of global transactions required to be undone or compensated as their write might not become permanent is,

$$N_{abort} = Py' \tag{9.68}$$

The number of exact global transactions required to be rolled back because they read the values written by the aborted transaction, is,

$$N_{roll} = o[1 - (1 - N_u/D)^{N_q}]y'N_{abort} \tag{9.69}$$

where $o$ is the fraction of exact global transactions.

Similarly, the cost of consistency restoration at the server SDB is calculated using the same formulas. The difference is that the conflicts between a global transaction, $GT$, that originates from a site and a local transaction, $LT$, from the server SDB site that the $GT$ is sent to. That is, a $GT$ is rolled back if its writes conflict with a read operation of a $LT$ that follows it in a GSG. Thus, assuming that reconciliation occurs after $N_r$ transactions, of which $z = (1-c)k_A fhNr$ are local and $z = (1-c)k_{A-1}fhNr$ are global. The size of the server SDB will now be $D^{server} = k_{A-1}D$. Formulas 9.53 to 9.67 can be used by substituting local for strict and $D^{server}$ for $D$, to determine

$$N_{abort}^{server} = Pz' \tag{9.70}$$

and

$$N_{roll}^{server} = o[1 - (1 - N_u/D^{server})^{N_q}]z'N_{abort}^{server} \tag{9.71}$$

## 9.3 Performance Evaluation

**Table 9.1. Input parameters**

| Parameter | Description | Value |
|---|---|---|
| $d$ | number of sites per server/central | 3 |
| $A$ | maximum number of tiers | 4 |
| $\lambda_q$ | query arrival rate | 12 queries/second |
| $\lambda_u$ | update arrival rate | 3 updates/second |
| $c$ | consistency factor | ranges from 0 to 1 |
| $f$ | hot spot factor | ranges from 0 to 1 |
| $h$ | fraction of database summarised | ranges from 0 to 1 |
| $k_A$ | coefficient of $h$ for $A$th tier local site | 0.5 |
| $k_{A-1}$ | coefficient of $h$ for $(A-1)$th tier server site | 0.7 |
| $t_q^C$ | processing time of a query at CDB | 0.00025 seconds |
| $t_u^C$ | processing time of a update at CDB | 0.001 seconds |
| $t_q^S$ | processing time of a query at server SDB | 0.0025 seconds |
| $t_u^S$ | processing time of a update at server SDB | 0.01 seconds |
| $t_q^L$ | processing time of a query at local SDB | 0.025 seconds |
| $t_u^L$ | processing time of a update at local SDB | 0.1 seconds |
| $t_b$ | propagation overhead | 0.00007 seconds |
| $V$ | vacation interval | ranges |
| $W$ | available bandwidth | ranges |
| $m$ | average size of a message | 512 bits |
| $D$ | average number of attribute values per site one tier away from the CDB | 1000 |
| $N$ | average number of operations per transaction | 10 |
| $T_c$ | average commitment time | 0.025 seconds |

### 9.3.1 Communication Cost

Communication costs are estimated by the number of messages sent. Using the parameters in Table 9.1, Figure 9.1 presents the average number of messages as a function of $h$ and $c$. As expected, when no strict transactions are present ($c = 0$), the average number of messages is dependent on how much of the database is stored within the SDB. Hence, the number of messages sent decreases as the replication becomes full ($h = 1$). When there is no replication ($h = 0$), all transactions are forwarded to the CDB for processing. Similarly, when all transactions are strict ($c = 1$), the number of messages remain constant, since all transactions are propagated to the CDB.

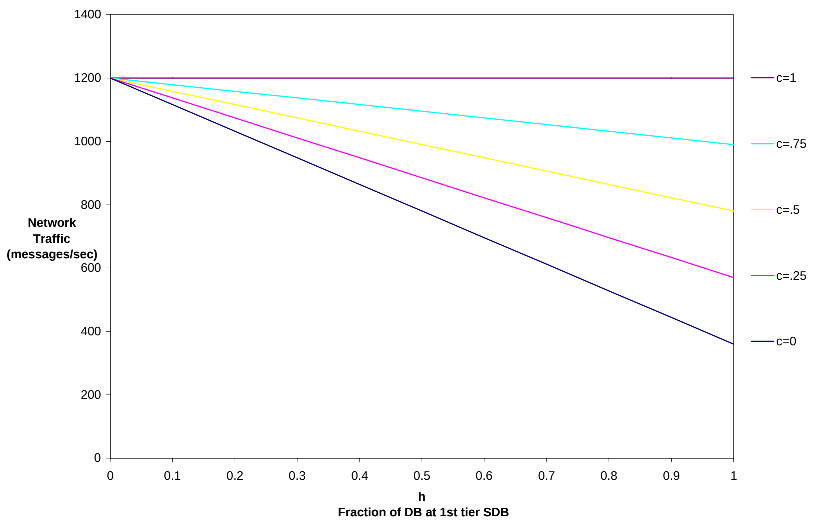


**Figure 9.1. Communication Cost**

### 9.3.2 Transaction Response Time

Using the parameters in Table 9.1, the response time for local transactions are given in Figure 9.2. The *left* figure depicts a response time given the locking method presented in Table 8.2, while the *right* uses the locking method shown in Table 8.3. As expected, the response time, *(left)*, increases as the percentage of the data existing in the local SDB increases. This occurs because the local SDB is required to process more local transactions, as it stores a larger percentage of the CDB. However, as the consistency requirement increases (the users submit more strict transactions), the response time decreases as there are less local transactions contending for data. In comparison, for the second locking method's *(right)*, increase in response time is more rapid, which is caused by conflicts occurring between different types of transactions.

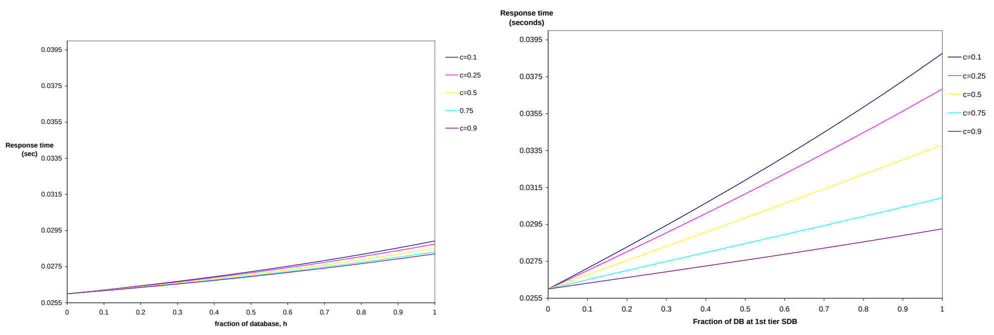


**Figure 9.2. Response time for Local Transactions**

The response time for global transactions are shown in Figure 9.3, where the *left* uses the locking method in Table 8.2 and the *right* uses the locking method in Table 8.3. The response time for the *left* figure depends on the required consistency, where for high values of $c$, the response time decreases as less global transactions are contending for the same data. While for lowest value of $c$, the response time increases as there is more contention for the same data.

While the *right* figure shows a more rapid decrease in response time for larger values of $c$, due to more transactions being processed locally, instead of being submitted as a global transaction, and much of the transactions being strict transactions. Therefore, less contention for the same data occurs. While for low value of $c$ an increase is evident as the local transactions of a server SDB conflict with the global transactions of a local SDB, one tier down.

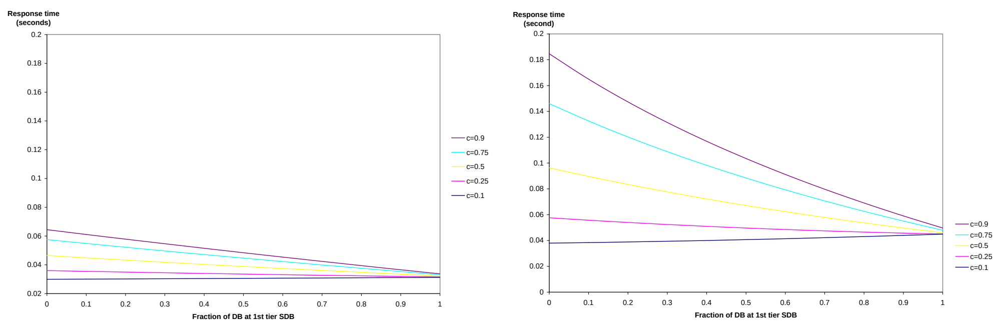


**Figure 9.3. Response time for Global Transactions**

Strict transactions require propagation to the CDB regardless of how much of the required data are stored locally. It is, therefore, more useful to examine response times in terms of consistency requirements. Figure 9.4 presents the response times of the two locking mechanisms for strict transactions, *Method 1* and *Method 2* (Tables 8.2 and 8.3 respectively), given various consistency value. This shows that as $c$ increases, response time increases as there are more conflicts occuring. Strict transactions of *Method 1* have a faster response time than *Method 2* since other transaction types do not contend for the same data. However,the average response time for strict transactions are, in general, much longer as they are propagated through several communication links to the CDB.

## 9.3.3 Reconciliation Costs

Given a size for the CDB, the cost of reconciliation is calculated using the parameters in Table 9.1. Figure 9.5 presents the probability of a global transaction not being accepted because of a conflict with a strict transaction, given that reconciliation occurred after a certain number of transactions and different consistencies.

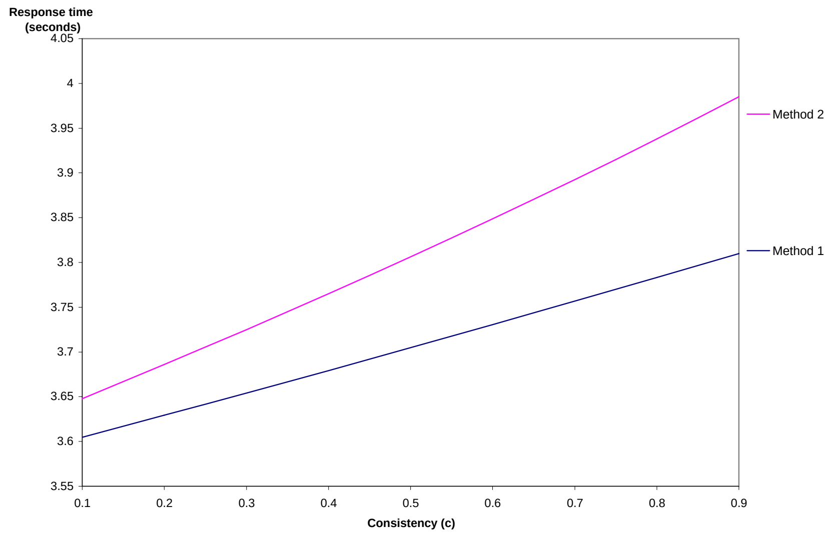


**Figure 9.4. Response time for Strict Transactions**

The *left*, and *right* figures, shows the probability of an abort caused at the CDB, and server SDB respectively.

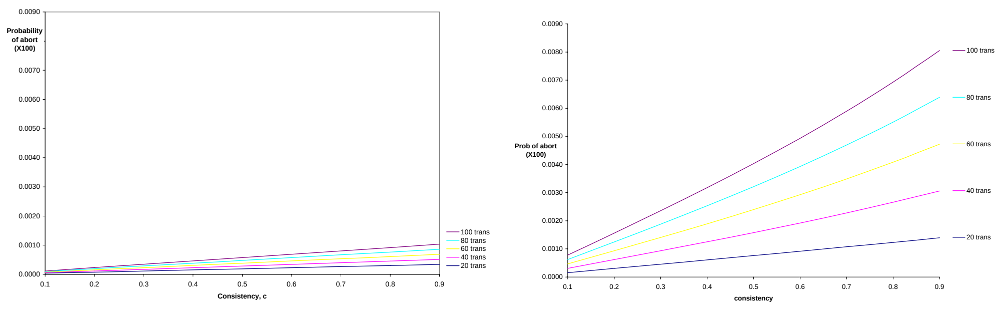


**Figure 9.5. Probability of abort for CDB and server SDB**

The figure indicates that there is a higher probability of aborts occurring at the server SDB, than at the CDB. The reason for this is that at a server SDB level, many global transactions are filtered out and aborted, due to conflicts, before they reach the CDB. Thus, reducing the number of transactions that are involved in an abort.

Furthermore, the figure illustrates that as a greater percentage of strict transactions are submitted, the probability of a global transaction being aborted increases. This is because strict transactions require more values from the database and thus, global transactions are blocked and aborted. These results are used to determine after how many transactions a reconciliation events should occur, to ensure that the probability of an abort is kept to a minimum.

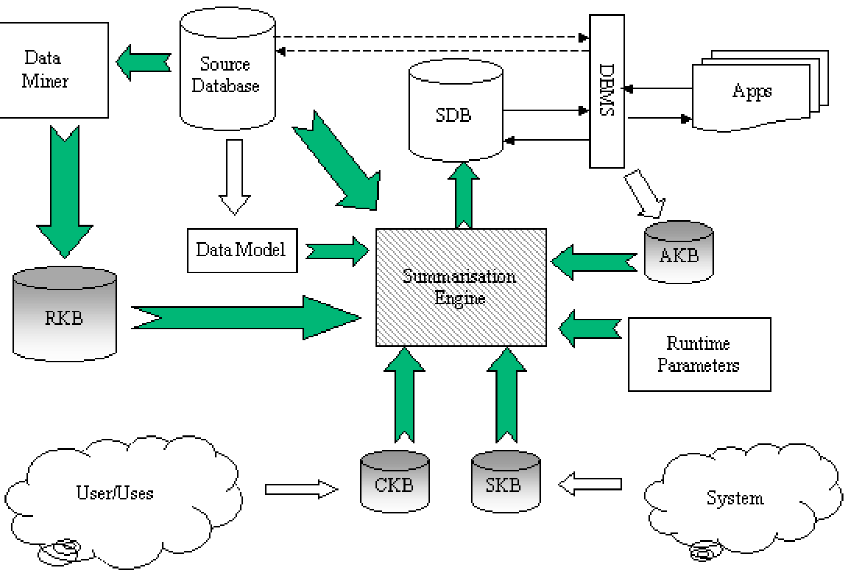


**Figure 9.6. Overview of the Summarisation Process**

## 9.4 System Simulation

### 9.4.1 Summarisation Engine Implementation

The main body of the summarisation engine combines the data gathered by the criteria to be used in the priority formula. Figures 5.2 and 9.6 provide an overview of the *Summarisation Engine*. The two databases, *Source Database* and *SDB*, represent the main database and the resulting summary database, located within the mobile unit. Through a DBMS, application programs access the summary and main database, depending on network connectivity.

The *Summarisation Engine* has inputs from seven sources:

- **The Source Database**
- **Knowledge Bases**
  - **Context Knowledge Base (CKB)**
    Consisting of user-supplied information, including a list of enumerated data of information concerning the users that can be used for contextual values. Either or both of which are optional.
  - **System Knowledge Base (SKB)**
    This is generated by the system and includes system information such as storage size, power requirements and communication network.

[†]This section appears as part of Chan and Roddick (2003)

  - **Rule Knowledge Base (RKB)**
    This contains rules generated by any external data mining engine.
  - **Application Knowledge Base (AKB)**
    This provides a feedback mechanism when deciding on data storage prioritisation. The AKB is generated by a DBMS to track requested data. The information is then used to include or delete data during the future recreation of summary database. For example, minimal used data can indicate a low degree of importance and hence, this data may be excluded in subsequent recreation.

- **Data Model**
  The schema/model of the database which is a simplified EER format, in this thesis.

- **Runtime Parameters**
  Include the specification of protocol as described in Table 5.2 and other information such as target location and SDB size.

Stage 2 of the summary database construction creates the database and reflects the priorities generated in Stage 1, such that the description length of the summary database is a small fraction of the total space available. In the case where all the primary keys are stored, the description length is a fixed value that depending on the database size. Alternatively, if only the required keys are stored, there is a trade-off between, including low priority data items and the costs of specifying the view structure. For the prototype, the former case has been chosen and the algorithm used by each criteria to select attribute values is given in Section 5.2.

The modular, plug-compatible implementation of the criteria means that modules representing, for example, the determination of $\phi$ for each of the criteria described earlier can either be developed specifically for the application or a generic module can be adopted. The prototype is then used to create the database required for the simulation implementation.

### 9.4.2 Simulation Implementation

A discrete-event simulation has been developed to validate the response time analysis given in Section 9.3. The simulation keeps track of the data accessed by each transaction and explicitly simulates data contention, transaction aborts, the locking of attribute values, locks availablility, queuing and processing at each site, I/O waits, communication delays and commit processing.

Sites that have consistently strong connection to each other are simulated as having a common work queue, while geographically distributed and mobile sites have separate queues. All transaction arrival processes are independent and Poisson. The service times at each site are no longer exponentially distributed as in the analytical model, but are constants that corresponds to the CPU MIPS rating.

Communication delays are modelled similarly to the vacation system model used in the analytical model. That is, the link is available with a probability of $p$ and the service times at each link are constants, plus the time required to wait for a link to become available, which is exponentially distributed. The commit times, I/O times and backoff times are all constants. The service handler at each site is released by a transaction when lock contention occurs, and during backoff after an abort.

The states of a transaction in the simulation are the same as described in Section 9.2.2. In the case of a local transaction, it is first queued at the corresponding site and the initial I/O times are added. At the start of each subsequent state, a request is made to the local concurrency control component. In which the lock table is checked, and if a contention is detected, the service handlers are released, and the transaction is enqueued on that lock. Transactions that access the same attribute values as the first run of that transaction. In the case of a contention based deadlock, the latest transaction is aborted, and all locks held are released and a backoff interval is started. The transaction is then enqueued to run after the backoff interval expires and no other transactions are running at that site. Any update propagation activities are modelled as the arrival of new global transactions with the same attribute values accessed.

In the case of global transactions, the destination and originating sites are identified. The transactions are then queued at the originating site. The start of each subsequent state requires communication with the destination site for the lock status of the required attribute value. The remainder of the transaction state is then handled similarly to a local transaction. A new global transaction then arrives at the destination to represent the propagation of the updates made by the global transaction. For sites that are only one link away from the central, a strict transaction arrival process is used instead of a global transaction.

For strict transactions, only the originating site is required. Since all strict transactions are processed at the central site, they are queued there. The total communication delay is determined by the number of links from the originating site and the wait time at each link. Each state is then handled similarly to local transactions but at the central site. After commitment, update propagation occurs, through the arrival of a global transaction at the central site with its destination pointing to all sites one link away.

For every transaction types, a destination stack is used, where the first originating site is pushed onto the stack first, followed by any subsequent destinations. If an abort occurs, the stack is used to determine where the transaction originated from. Transaction generators used to input transactions into each site and are used to define the percentage of strict transactions introduced into the system.

## 9.4.3 Simulation Results

The average response time of local transactions are calculated using the simulation schedule as the average time taken to reach the queue, until the time they leave the system. The percentage of strict transactions, or consistency, are varied as with the analytical model, and all other parameters are in accordance to Table 9.1 unless otherwise specified. Figure 9.7 shows the average response times of local transactions versus the percentage of the database summarised at the tier-1 database, using the two locking schemes shown in Tables 8.2 and 8.3. Both the analytical approximations and simulation estimates are provided, where the solid line shows the former and the dash line shows the latter. Figure 9.8 shows the average response times for global transactions. While for strict transactions, presented in Figure 9.9, shows the average response times versus consistency, as a summarised database has no effect on strict transactions.

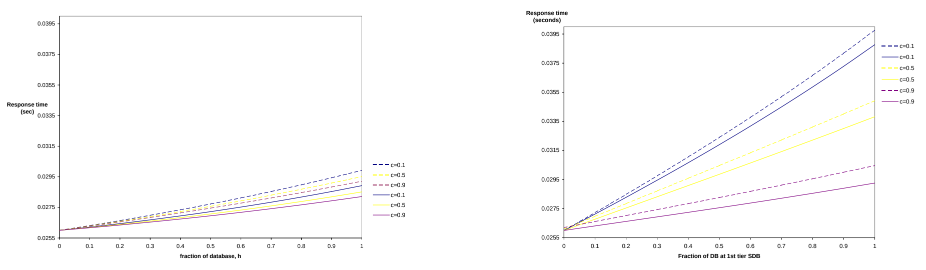


**Figure 9.7. Response time for Local Transactions**

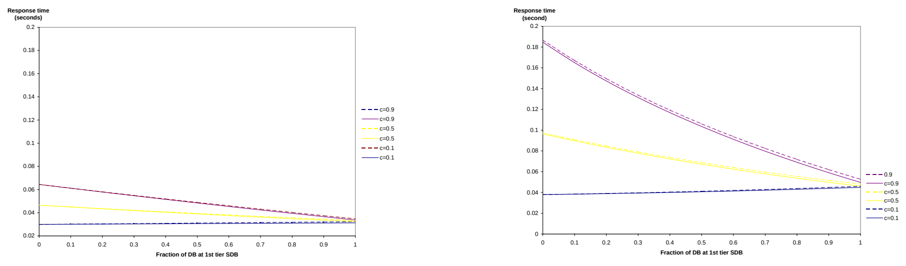


**Figure 9.8. Response time for Global Transactions**

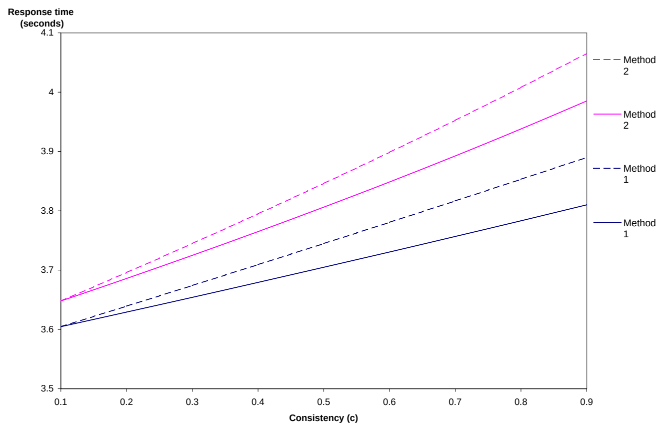


**Figure 9.9. Response time for Strict Transactions**

In all cases, there is good agreement between the simulation and the analysis. They are all close, especially in terms of the shapes of their curves.

## 9.5 Discussion

The COSMOS database system was evaluated in terms of its transaction processing, using a performance model to measure the response times of transactions within the COSMOS database system. An analytical approach was used for this evaluation, while a simulation was used to validate these analytical results. The evaluation shows that the use of local and global transactions increase response time as they are dependant on databases that are close to the originating transaction site. Strict transactions, on the other hand, increase the consistency between databases from different sites, but at the cost of response time when only weak connectivity is available. Thus, the use of strict transactions should be limited to periods when strong connectivity is available, or when the user requires accurate transaction results.

# Chapter 10

# Conclusion

This dissertation presents the development of COSMOS, a framework for context sensitive summarisation of a mobile distributed relational database. This chapter presents the contributions of this dissertation in Section 10.1, while further work in this area is discussed in Section 10.2

## 10.1 Contributions

The main contribution made by this dissertation is the creation of a framework for summarising and allocating data in a hierarchical mobile distributed database system, COntext Sensitive MObile Summarisation (COSMOS). Associated with the use of COSMOS are database and transaction management issues such as missing values, consistencies between databases and update protocols.

The creation of COSMOS consist of two stages. The first stage identifies important information with regards to the user by through the specification of criteria. These criteria can consist of:

- External enumeration of data.
- Contextual information on the user and the user's environment.
- Previous user's history.
- Push-based criterion that allows the main server to specify data importance.
- Model-based criterion that employs the database schema and its corresponding ER diagram.
- Inductive criterion that employs data mining techniques.

- Time-based criterion to filter out old data.
- Spatial-based criterion that uses location-awareness to filter out less important information.

The second stage then creates Storage Map (SM), which identify attribute values that are to be stored within the summarised mobile database.

Since a summary database is always a subset of a main database, there are always attribute values that are null in the summary, but have a value within the main database. From the summary database, it is quite difficult to determine whether they are global nulls without consulting the main database. To distinguish between the two, local nulls are used. Local nulls are defined as locally unavailable values, they extend the definitions of global nulls, but are limited to only summary databases. Local nulls allow the accuracy of a query to be varied according to the need of the user and the bandwidth available.

Transaction management within a COSMOS system requires modification to the ACID. In COSMOS, there are three types of transactions, local, global and strict. Local transactions are processed at the originating local SDB. Global transactions are handed over by the originating site to its server SDB transaction processing. Finally, strict transactions are processed at the CDB.

With the proposal of new transaction operations, a relaxation of consistency was required and hence proposed between databases with weak connections. For strongly connected databases, strict consistency is required. Bounded inconsistency is used to measure the amount of consistency between the databases. Reconciliation is therefore required after a number of transaction to ensure consistency between participating databases. The method proposed to reconcile attribute values uses a syntactic approach that employs serialisability-based criteria. To reconcile, the CDB examines whether transactions committed from lower tiers will successfully commit at the CDB. If unsuccessful, an abort occurs that rolls-back all transactions committed down the branch that the transaction originates from.

To ensure that COSMOS database system functions correctly, a list of protocols is required, including:

1. A function to transform the transaction operations to Storage Map (SM). This function uses transaction operations to map which attribute the operation is being applied to.

2. Protocols that ensure serialisability, by using a locking mechanism. Every transaction is required to check the locking table for each attribute value before it may operate on it. A local transaction checks locks at the local SDB. A global transaction checks locks from its server SDB, while strict transactions checks locks at the CDB.

3. Reconciliation protocols, which check the Global Schedule Graph (GSG) for cycles before either committing or aborting a transaction.

4. Protocols required when an update is submitted to a local summary database including the Local Update Propagation (LUP) and Update through local item invisibility. The former specifies that a local transaction can locally commit before propagating the result to the server SDB or CDB. While the latter, specifies that any updates that do not have all the required attribute values available locally, are required to 'mask' the available attribute values as invisible, and forward the updates to the server SDB.

5. Protocols required when commit phase at the central database is complete. Once committed database refresh procedures propagate updates to all pertinentdatabases. A push refresh allows these new values to be propagated to all the databases. A full refresh allows all the attribute values in a SDB to be updated and its importance to the user be recalculated. Finally, a pull refresh is also available to allow an SDB to update data accordingly to any change in user context.

6. Recovery protocols are required when the central database is inaccessible. Using an election protocol, the highest tier databases elect a primary database to act as the CDB until the actual CDB becomes accessible. Multiple primary databases are allowed, to coordinate the storage of different attribute values.

An evaluation was conducted using models based upon queuing systems. Analytical approximations and simulation estimates were used to determine the response times for different transactions. It showed that the use of local and global transactions reduce transaction response time. While the use of strict transactions requires strong connectivity between the originating SDB and the CDB. Otherwise, higher response time will be experienced because of the communication delays of wireless and long distanced networks.

## 10.2 Future Work

A natural extension of this dissertation is the determination of the optimal combination of summarisation criteria. While there may not be a best combination for every users, a group of users may share similar interest and hence, the same data. A survey of users may be required to properly determine the optimal set of data different groups of users might use. Further refinement by individual users may be required depending on the available storage. best combination.

Another possible extension to the COSMOS database system that is of interest to enable mobile databases to roam around different sub-networks. Roaming capabilities would allow a user to have access to any required information regardless of where they are. Currently in COSMOS, mobile databases can freely roam between sub-networks that contain a server database of which it is a subset. To roam other sub-networks would cause the server database to store all the data that is contained within the mobile database. This may not be practical since a mobile database may just be passing or only staying for short periods, but the server database would still be required to store all those data. A solution to this involves introducing proxies or middleware into the architecture, which can then act as caches for any mobile databases, storing only the required data temporarily.

Finally, allowing queries to return approximate results, instead of simply results with local nulls may decrease the need of local databases to access the rest of the network. One way to achieve this for databases, in particular the mobile databases, to be aware of the context used to create it. That is, if a criterion specifies that attribute values with the same value are to be included into the SDB, then that knowledge can be used to approximate the values of the local nulls for that attribute. For example, assume a criterion stores all values $a$ for an attribute, *COL1*. Then, a query searching for all $a$ in *COL1* will only be required to search the local SDB since it knows that any local nulls which result will not be an $a$ value.

# Appendix A

# Acronyms and Terminology

## A.1 Acronyms

| | |
|---|---|
| **3G** | Third Generation Mobile Networks |
| **A** | Abort for transactions |
| **ARM** | Association Rule Mining |
| **C** | Commit for transactions |
| **CDB** | Central Database |
| **CDMA** | Code Division Mutiple Access |
| **COSMOS** | COntext Sensitive MObile Summarisation |
| **DBMS** | Database Management System |
| **DWT** | Discreet Wavelet Transform |
| **GPS** | Global Positioning System |
| **GR** | Global Read |
| **GS** | Global Schedule for transactions |
| **GSG** | Global Schedule Graph for transactions |
| **GSM** | Global System for Mobile |
| **GW** | Global Write |

| | |
|---|---|
| **LAN** | Local Area Network |
| **LCD** | Liquid Crystal Display |
| **LR** | Local Read |
| **LS** | Local Schedule for transactions |
| **LSG** | Local Serialisation Graph for transactions |
| **LUP** | Local Update Propagation |
| **LW** | Local Write |
| **O2PL** | Optimistic Two Phase Locking |
| **O2PL-MT** | Optimistic Two Phase Locking for Mobile Transactions |
| **PC** | Personal Computer |
| **PDA** | Personal Digital Assistant |
| **SDB** | Summary Database |
| **SM** | Storage Map |
| **SR** | Strict Read |
| **SVD** | Singular Value Decomposition |
| **SW** | Strict Write |
| **UDB** | Update Database |
| **WiFi** | Wireless Fidelity |
| **WLAN** | Wireless Local Area Network |
| **WPAN** | Wireless Personal Area Network |
| **WWAN** | Wireless Wide Area Network |

## A.2 Terminology

**Attribute Value** are the actual values stored for in a database for each corresponding tuple and attribute.

**CDB** is the central database which stores all information available to a database system.

**Local SDB** are client sites which may not act as a server for any other sites.

**Server SDB** are client sites which may act as a server for any other sites.

# Appendix B

# Example

This example, assumes the case of a general hospital, in which many users access the hospital's database. For this example, however, only one user will be examined here. The database used has a schema given in Figure 5.3. The specific criteria used to create the summary database for the user is given in this appendix.

The first user examined is a doctor working at the hospital. In this example, Table B.1 provides the criteria used and the relative importance of each criteria. The following list provides a description of how each criterion is used.

**Table B.1. The setting of $\rho$ for each data inclusion criterion.**

| Criteria | | $\rho$ |
|---|---|---|
| Enumerated | $e$ | 100 |
| Contextual | $c$ | 75 |
| Previous Usage | $u$ | 30 |
| Push-based | $p$ | 30 |
| Model-based | $m$ | 90 |
| Inductive | $i$ | 60 |
| Time-based | $t$ | 0 |
| Spatially-based | $s$ | 20 |

Since previous treatments are still valid and possibly required for current treatment, the use of time-based criterion is not required for this user.

- **Enumerated**

  For this criterion, the doctor had earlier pre-enumerated a list of today's patients. This list is used to generate a list of all patient's code for the corresponding patient, Table B.2.

  The enumerated list has the highest relative priority because it contains critical information. The assigned priority, $\phi$, given to list enumerated is 1 to indicate that they are to be included.

**Table B.2. Enumerated patient's list.**

| *patCode* |
|---|
| 1002 |
| 1013 |
| 1040 |
| 1060 |

- **Contextual**

  Given that the doctor is an ear, nose and throat specialist, a contextual search of the database discovers that the admission records in Table B.3*(left)* had treatments for related disorders. Of the list patients, Table B.3*(centre)* shows the a list of patients that have visited this doctor. While Table B.3*(right)* shows the ones that have not.

**Table B.3. Contextual patient's list.**

| *admCode* |
|---|
| 3001002 |
| 3001013 |
| 3001020 |
| 3001024 |
| 3001040 |
| 3001050 |
| 3001055 |
| 3001060 |
| 3001200 |
| 3001201 |

| *admCode* |
|---|
| 3001002 |
| 3001013 |
| 3001024 |
| 3001040 |
| 3001060 |
| 3001201 |

| *admCode* |
|---|
| 3001020 |
| 3001050 |
| 3001055 |
| 3001200 |

  The contextual information has been allocated the second highest relative priority since it contains information the doctor will need if any patients turn up without an appointment. The assigned priority, $\phi$, is given a value of 1 for the list in B.3*(centre)*, and a 0.5 for the list in B.3*(right)*. This is because the centre list contains patients that are more likely to visit the doctor than the ones on the right.

- **Previous Usage**

  For this criterion, assume that with patient with a *patCode* of 1024 has weekly appointments with the doctor, 1002 has fortnightly appointments, and 1201 has monthly appointments. For each patient, the frequency that the doctor accesses their details correlate to appointment frequency. Thus, the assigned priority, $\phi$, may be given the values of 1, 0.7 and 0.5 for patients with weekly, fortnightly and monthly appointments, respectively.

- **Push-based**

  The server or the main database specifies what information is pushed to its clients. The information are general notices that affect all the hospital staff, includes system downtimes or any intending upgrades to the system. Assuming a short system upgrade has been planned for the afternoon. Since the database schema used does not provide for system notices, an email system has been used to notify the staff. The specified email has an assigned priority of 1 and is installed into the calendar doctor's mobile device as a reminder that the server is unavailable.

- **Model-based**

  Using the data collected by the primary criteria (Enumerated, Contextual and Previous Usage) and the database schema, new information is collected by this criteria. The first data identified are the tuples corresponding to data selected by primary criteria. In the case of the *Patient* table the tuples identified are shown Table B.4.

**Table B.4.** ***Patient*** **Table**

| *patCode* | **name** | **sex** | **age** | **town** | **physician** |
|---|---|---|---|---|---|
| *1002* | Jay Bedford | M | 44 | Adelaide | 9001 |
| *1013* | Clara Hall | F | 21 | Adelaide | 9001 |
| *1024* | Julie Long | F | 23 | Adelaide | 9001 |
| *1040* | Hector Best | M | 25 | Adelaide | 9001 |
| *1060* | Michael Biggs | M | 53 | Adelaide | 9001 |
| *1201* | Andrew Perez | M | 40 | Adelaide | 9001 |

Using an iterative method, the second step then identifies all data in relationship tables that are one link away. This is done using the schema's corresponding ER diagram. The tables *Town*, *Physician* and *Admission* are all one link away from the *Patient* table. The third step identifies all data two links away, and so on. The continues until all tables have been searched, or until a specified search depth has been reached. Given a search depth of two, the third step identifies data from *Diagnosis*, *D.R.G*, *Ward*, *Surgery*. Notice that *Physician* and *Admission* tables are only counted once since only the shortest distance is used.

To calculate the assigned priority, $\phi$, the shortest distance between the new data and the original data collected by Primary criteria is used in Formula 5.3. That is, $\phi_m = k^{-a}$, where $k = 2$ and $a$ is the shortest distance given by the ER diagram. The following tables (Table B.5) show the data collected by the second step.

The process is repeated for attribute values in other tables selected by all primary criteria, such as those selected in the *Admission* table.

**Table B.5.** *Town*, *Physician*, *Admission* **Tables**

| *town* |
|---|
| *Adelaide* |
| *Sydney* |

| *physician* |
|---|
| *9001* |

| *admCode* | **patCode** | **date** | **n.days** | **outcome** | **diagnosis** |
|---|---|---|---|---|---|
| *3001002* | 1002 | 12 Jan 2003 | 3 | still in hospital treatment | ear implant |
| *3001013* | 1013 | 24 Feb 2003 | 2 | still in hospital treatment | sinus |
| *3001020* | 1020 | 15 Oct 2004 | NULL | NULL | NULL |
| *3001024* | 1024 | 2 Dec 2001 | 3 | still in hospital treatment | ear infection |
| *3001040* | 1040 | 18 Mar 2002 | 2 | still in hospital treatment | flu |
| *3001050* | 1050 | 22 Nov 2003 | 3 | NULL | sinus |
| *3001060* | 1060 | 10 Aug 2003 | 3 | still in hospital treatment | fever |
| *3001200* | 1200 | 26 Sep 2004 | 1 | prescribed antibiotics | allergies |
| *3001201* | 1201 | 7 May 2002 | 1 | prescribed antibiotics | sorethroat |

- **Inductive**
  Assuming the Apriori algorithm is used and inferences have been identified for the main database, some of the rules that are generated are given in the following list:
  - flu $\rightarrow$ fever (66%)
  - flu $\rightarrow$ sorethroat (60%)
  - flu $\rightarrow$ antibiotic (40%)
  - fever $\rightarrow$ headache (66%)
  - fever $\rightarrow$ sorethroat (100%)
  - fever $\rightarrow$ antibiotics (100%)

  The easiest way to determine the assigned priority, $\phi$, is to equate it to the confidence of each rule. For this example, a list of admission codes are generated from the rules corresponding with the admissions given in Table B.5 (*bottom*) and given in Table B.6.

**Table B.6.** *Admission* **Table after Inductive criterion used.**

| *admCode* | **patCode** | **date** | **n.days** | **outcome** | **diagnosis** |
|---|---|---|---|---|---|
| *3001003* | 1003 | 2 Jan 2003 | 1 | prescribed antibiotics | sorethroat |
| *3001023* | 1023 | 22 Feb 2003 | 1 | prescribed antibiotics | NULL |
| *3001036* | 1036 | 15 Oct 2003 | 1 | prescribed antibiotics | headache |
| *3001060* | 1060 | 10 Aug 2003 | 1 | still in hospital treatment | fever |
| ... | ... | ... | ... | ... | ... |

- **Spatially-based**
  Since the hospital is located in Adelaide, there is a less probability that patients from a different town will revisit. Thus, patients from Adelaide selected by primary and secondary criteria will be given a higher assigned

priority, $\phi$, value than those from another state. That is, $\phi = 1$ when town = 'Adelaide' and $\phi = 0.2$ when town = 'Sydney'.

All priorities calculated by each criterion are summed together using Formula 5.1. All attribute values have a corresponding priority weighting assigned to it. Table B.7 and B.8 show partial *Patient* and *Admission* tables with corresponding priority weightings.

**Table B.7.** ***Patient*** **Table**

| *patCode* | **name** | **sex** | **age** | **town** | **physician** |
|---|---|---|---|---|---|
| *1002* | Jay Bedford | M | 44 | Adelaide | 9001 |
| *1013* | Clara Hall | F | 21 | Adelaide | 9001 |
| *1020* | Michele Moore | F | 39 | Adelaide | 9001 |
| *1024* | Julie Long | F | 23 | Adelaide | 9001 |
| *1040* | Hector Best | M | 25 | Adelaide | 9001 |
| *1050* | Jay Doe | M | 30 | Sydney | 9001 |
| *1060* | Michael Biggs | M | 53 | Adelaide | 9001 |
| *1200* | Tim Cutten | M | 30 | Adelaide | 9001 |
| *1201* | Andrew Perez | M | 40 | Adelaide | 9001 |

| *patCode* | **name** | **sex** | **age** | **town** | **physician** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| *1002* | 21.1 | 211 | 70.33 | 26.37 | 52.75 |
| *1013* | 19 | 190 | 63.33 | 23.75 | 47.5 |
| *1020* | 9 | 90 | 30 | 11.25 | 22.5 |
| *1024* | 12 | 120 | 40 | 15 | 30 |
| *1040* | 19 | 190 | 63.33 | 23.75 | 47.5 |
| *1050* | 9 | 90 | 30 | 11.25 | 22.5 |
| *1060* | 19 | 190 | 63.33 | 23.75 | 47.5 |
| *1200* | 9 | 90 | 30 | 11.25 | 22.5 |
| *1201* | 10.5 | 105 | 35 | 13.12 | 26.25 |
| ... | ... | ... | ... | ... | ... |

Assuming that the database is capable of storing any attribute value which has a priority weighting higher than 8, then, Table B.9 shows the *Patient* and *Admission* tables stored in the user's SDB. Note that for the *Patient* table, more attribute values are stored since it is much small in size than the *Admission* table.

**Table B.8.** ***Admission*** **Tables**

| *admCode* | **patCode** | **date** | **n.days** | **outcome** | **diagnosis** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| *3001002* | 1002 | 12 Jan 2003 | 3 | still in hospital treatment | ear implant |
| *3001003* | 1003 | 2 Jan 2003 | 1 | prescribed antibiotics | sorethroat |
| *3001013* | 1013 | 24 Feb 2003 | 2 | still in hospital treatment | sinus |
| *3001020* | 1020 | 15 Oct 2004 | NULL | NULL | NULL |
| *3001023* | 1023 | 22 Feb 2003 | 1 | prescribed antibiotics | NULL |
| *3001024* | 1024 | 2 Dec 2001 | 3 | still in hospital treatment | ear infection |
| *3001036* | 1036 | 15 Oct 2003 | 1 | prescribed antibiotics | headache |
| *3001040* | 1040 | 18 Mar 2002 | 2 | still in hospital treatment | flu |
| *3001050* | 1050 | 22 Nov 2003 | 3 | NULL | sinus |
| *3001055* | 1055 | 30 Jul 2004 | NULL | NULL | NULL |
| *3001060* | 1060 | 10 Aug 2003 | 3 | still in hospital treatment | fever |
| *3001200* | 1200 | 26 Sep 2004 | 1 | prescribed antibiotics | allergies |
| *3001201* | 1201 | 7 May 2002 | 1 | prescribed antibiotics | sorethroat |
| ... | ... | ... | ... | ... | ... |

| *admCode* | **patCode** | **date** | **n.days** | **outcome** | **diagnosis** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| *3001002* | 41.25 | 15 | 41.25 | 8.25 | 16.5 |
| *3001003* | 9 | 3.27 | 9 | 1.8 | 3.6 |
| *3001013* | 41.25 | 15 | 41.25 | 8.25 | 16.5 |
| *3001020* | 9.375 | 3.40 | 9.375 | 1.87 | 3.75 |
| *3001023* | 6 | 2.18 | 6 | 1.2 | 2.4 |
| *3001024* | 41.25 | 15 | 41.25 | 8.25 | 16.5 |
| *3001036* | 9.9 | 3.6 | 9.9 | 1.98 | 3.96 |
| *3001040* | 41.25 | 15 | 41.25 | 8.25 | 16.5 |
| *3001050* | 9.375 | 3.40 | 9.37 | 1.87 | 3.75 |
| *3001055* | 9.9 | 3.6 | 9.9 | 1.98 | 3.96 |
| *3001060* | 41.25 | 15 | 41.25 | 8.25 | 16.5 |
| *3001200* | 9.37 | 3.40 | 9.37 | 1.87 | 3.75 |
| *3001201* | 41.25 | 15 | 41.25 | 8.25 | 16.5 |

**Table B.9.** ***Patient*** **and** ***Admission*** **Table**

| *patCode* | **name** | **sex** | **age** | **town** | **physician** |
|---|---|---|---|---|---|
| *1002* | Jay Bedford | M | 44 | Adelaide | 9001 |
| *1013* | Clara Hall | F | 21 | Adelaide | 9001 |
| *1020* | Michele Moore | F | 39 | Adelaide | 9001 |
| *1024* | Julie Long | F | 23 | Adelaide | 9001 |
| *1040* | Hector Best | M | 25 | Adelaide | 9001 |
| *1050* | Jay Doe | M | 30 | Sydney | 9001 |
| *1060* | Michael Biggs | M | 53 | Adelaide | 9001 |
| *1200* | Tim Cutten | M | 30 | Adelaide | 9001 |
| *1201* | Andrew Perez | M | 40 | Adelaide | 9001 |

| *admCode* | **patCode** | **date** | **n.days** | **outcome** | **diagnosis** |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| *3001002* | 1002 | 12 Jan 2003 | 3 | still in hospital treatment | ear implant |
| *3001003* | 1003 | LNULL | 1 | LNULL | LNULL |
| *3001013* | 1013 | 24 Feb 2003 | 2 | still in hospital treatment | sinus |
| *3001020* | 1020 | LNULL | NULL | NULL | NULL |
| *3001023* | LNULL | LNULL | LNULL | LNULL | NULL |
| *3001024* | 1024 | 2 Dec 2001 | 3 | still in hospital treatment | ear infection |
| *3001036* | 1036 | LNULL | 1 | LNULL | LNULL |
| *3001040* | 1040 | 18 Mar 2002 | 2 | still in hospital treatment | flu |
| *3001050* | 1050 | LNULL | 3 | NULL | LNULL |
| *3001055* | 1055 | LNULL | NULL | NULL | NULL |
| *3001060* | 1060 | 10 Aug 2003 | 3 | still in hospital treatment | fever |
| *3001200* | 1200 | LNULL | 1 | LNULL | LNULL |
| *3001201* | 1201 | 7 May 2002 | 1 | prescribed antibiotics | sorethroat |
| ... | ... | ... | ... | ... | ... |

# Bibliography

(n.d.).

Abowd, G. D., Atkeson, C., Hong, J., Long, S., Kooper, R. & Pinkerton, M. (1997), 'Cyberguide: A mobile context-aware tour guide', *ACM Wireless Networks* **3**, 421–433.

Aggarwal, A., Kapoor, M., Ramachandran, L. & Sarkar, A. (2000), Clustering algorithms for wireless ad hoc networks, *in* '4th International Workshop on Discreete Algorithms and Methods for Mobile Computing and Communications', Boston, MA, pp. 54–63.

Agrawal, R., Imielinski, T. & Swami, A. (1993), Mining association rules between sets of items in large databases, *in* P. Buneman & S. Jajodia, eds, 'ACM SIGMOD International Conference on the Management of Data', ACM Press, Washington DC, USA, pp. 207–216.

Ahmad, I., Karlapalem, K., Kwok, Y.-K. & So, S.-K. (2002), 'Evolutionary algorithms for allocating data in distributed database systems', *Distributed and Parallel Databases* **11**(1), 5–32.

Al-Mogren, A. S. & Dunham, M. H. (2001), Buc, a simple yet efficient concurrency control technique for mobile data broadcast environment, *in* '12th International Workshop on Database and Expert Systems Applications (DEXA 2001)', IEEE Computer Society, Munich, Germany, pp. 564–569.

Alonso, R., Barbara, D. & Garcia-Molina, H. (1990), 'Data caching issues in an information retrieval system', *ACM Transactions on Database Systems* **15**(3), 359–384.

ANSI/X3/SPARC (1975), 'Interim report: ANSI/X3/SPARC Study Group on Data Base Management Systems 75-02-08', *FDT - Bulletin of ACM SIGMOD* **7**(2), 1–140.

Antoshenkov, G. (1993), 'Query processing in DEC RDB: Major issues and future challenges', *IEEE Data Engineering Bulletin* **16**(4), 42–45.

Aoki, P. (1998), Generalizing "search" in generalized search trees, *in* '14th International Conference on Data Engineering', Orlando, FL, USA.

Barbara, D., DuMouchel, W., Faloutsos, C., Haas, P. J., Hellerstein, J. M., Ioannidis, Y. E., Jagadish, H., Johnson, T., Ng, R., Poosala, V., Ross, K. A. & Sevcik, K. C. (1997), 'The New Jersey data reduction report', *IEEE Data Engineering Bulletin: Special Issue on Data Reduction Techniques* **20**(4), 3–45.

Barbara, D. & Garcia-Molina, H. (1982), How expensive is data replication? an example, *in* '2nd International Conference on Distributed Computer Systems', pp. 263–268.

Bardram, J., Kjaer, R. & Pedersen, M. (2003), Context-aware user authentication-supporting proximity-based login in pervasive computing, *in* 'Ubicomp 2003', pp. 107–123.

Bayer, R. & McCreight, E. M. (1972), 'Organization and maintenance of large ordered indices', *Acta Informatica* pp. 173–189.

Bell, T., Witten, I. H. & Cleary, J. G. (1989), 'Modeling for text compression', *ACM Computing Surveys* **21**(4), 557–591.

Bernstein, P. A., Hadzilacos, V. & Goodman, N. (1987), *Concurrency Control and Recovery in Database Systems*, Addison-Wesley.

Biskup, J. (1983), 'A foundation of codd's relational maybe-operations', *ACM Transactions on Database Systems* **8**(4), 608–636.

Breitbart, Y., Garcia-Molina, H. & Silberschatz, A. (1992), 'Overview of multidatabase transaction management', *VLDB Journal: Very Large Data Bases* **1**(2), 181–293.

Breitbart, Y., Garcia-Molina, H. & Silberschatz, A. (1995), Transaction management in multidatabase systems., *in* 'Modern Database Systems', pp. 573–591.

Breitbart, Y., Komondoor, R., Rastogi, R., Seshadri, S. & Silberschatz, A. (1999), Update propagation protocols for replicated databases, *in* 'ACM SIGMOD International Conference on the Management of Data', Philadelphia, PA, USA, pp. 97–108.

Brown, P. J. (1996), The stick-e document: A framework for creating context-aware applications, *in* 'Electronic Publishing '96', pp. 259–272.

Brown, P. J., Bovey, J. & Chen, X. (1997), 'Context-aware applications: From the laboratory to the marketplace', *ACM Wireless Networks* **3**, 421–433.

Brunstrom, A., Leutenegger, S. T. & Simha, R. (1995), Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workloads, *in* '4th International Conference on Information and Knowledge Management, CIKM'95'.

Cannane, A. & Williams, H. E. (2001), 'General-purpose compression for efficient retrieval', *Journal of the American Society for Information Science and Technology* **52**(5), 430–437.

Cannane, A., Williams, H. E. & Zobel, J. (1999), A general-purpose compression scheme for databases, *in* 'Data Compression Conference', p. 519.

Carey, M. J. & Livny, M. (1991), 'Conflict detection tradeoffs for replicated data', *ACM Transactions on Database Systems* **16**(4), 703–746.

Ceri, S., Negri, M. & Pelagatti, G. (1982), Horizontal data partitioning in database design, *in* 'ACM SIGMOD International Conference on the Management of Data', ACM Press, Orlando, FL, USA, pp. 128–136.

Chalmers, M. (2004), 'A historical view of context', *Computer Supported Coperative Work* **13**(3-4), 223–247.

Chan, D. & Roddick, J. F. (2003), Context-sensitive mobile database summarisation, *in* M. Oudshoorn, ed., '26th Australasian Computer Science Conference (ACSC2003)', Vol. 16 of *CRPIT*, ACS, Adelaide, Australia, pp. 139–149.

Chan, D. & Roddick, J. F. (2005), 'Summarisation for mobile databases', *Journal of Research and Practice in Information Technology* **37**(3), 267–284.

Chan, D. & Roddick, J. F. (2006), Local nulls in summarised mobile and distributed databases, *in* 'Data Mining and Information Engineering 2006', Vol. 37 of *WIT Transactions on Information and Communication Technologies*, WIT Press.

Chen, S.-M. & Chen, H.-H. (2000), 'Estimating null values in the distributed relational databases environment', *Cybernetics and Systems* **31**(8), 851–871.

Cheung, S., Ammar, M. & Ahamad, M. (1992), 'The grid protocol: A high performance scheme for maintaining replicated data', *IEEE Transactions on Knowledge and Data Engineering* **4**(6), 582–592.

Christensen, H. & Bardram, J. (2002), Supporting human activities - exploring activity-centered computing, *in* 'Ubicomp 2002', Göteborg, pp. 107–116.

Chrysanthis, P. K. (1993), Transaction processing in a mobile computing environment, *in* 'IEEE workshop on Advances in Parallel Distributed System', pp. 77–82.

Chu, P.-C. (1992), 'A transaction-oriented approach to attribute partitioning', *Information Systems* **17**(4), 329–342.

Ciciani, B., Dias, D. M. & Yu, P. S. (1990), 'Analysis of replication in distributed database systems', *IEEE Transactions on Knowledge and Data Engineering* **2**(2), 247–261.

Cochran, W. (1977), *Sampling Techniques*, 3rd edn, Wiley, New York.

Codd, E. (1970), 'A relational model of data for large shared data banks', *Communications of the ACM* **13**(6), 377–387.

Codd, E. (1975), 'Understanding relations (installment #7)', *FDT-Bulletin of ACM SIGMOD* **7**(3), 23–28.

Codd, E. (1979), 'Extending the database relational model to capture more meaning', *ACM Transactions on Database Systems* **4**(4), 397–434. An early version of this work was presented at the Second Australian Computer Science Conference in Hobart, Tasmania, Australia.

Codd, E. (1986), 'Missing information (applicable and inapplicable) in relational databases', *SIGMOD Record* **15**(4).

Codd, E. (1987), 'More commentary on missing information in relational databases (applicable and inapplicable information)', *SIGMOD Record* **16**(1).

Comer, D. (1979), 'The ubiquitous b-tree', *Computing Surveys* **11**(2), 121–137.

Connolly, T., Begg, C. & Strachan, A. (1999), *Database Systems: A Practical Approach to Design, Implementation and Management*, 2nd edn, Addison-Wesley, Harlow.

Cormack, G. V. (1985), 'Data compression on a database system', *Communications of the ACM* **28**(12), 1336–1342.

Cornell, D. W. & Yu, P. S. (1990), A vertical partitioning algorithm for relational databases, *in* 'Third International Conference on Data Engineering, ICDE', pp. 30–35.

Covington, M. J., Long, W., Srinivasan, S., Dev, A. K., Ahamad, M. & Abowd, G. D. (2001), Securing context-aware applications using environment roles, *in* 'Proceedings of the Sixth ACM symposium on Access control models and technologies', ACM Press, Chantilly, Virginia, US, pp. 10–20.

Cuce, S., Zaslavsky, A. B., Hu, B. & Rambhia, J. (2002), Maintaining consistency of twin transaction model using mobility-enabled distributed file system environment, *in* '13th International Workshop on Database and Expert Systems Applications (DEXA 2002)', Aix-en-Provence, France, pp. 752–756.

Date, C. (1986), Null values in database management, *in* 'Relational Database: Selected Writings', Addison-Wesley, Reading, MA.

Daubechies, I. (1992), 'Ten lectures on wavelets', *Society for Indusrial and Applied Mathematics (SIAM)* .

Davidson, S. B., Garcia-Molina, H. & Skeen, D. (1985), 'Consistency in partitioned networks', *ACM Computing Surveys* **17**(3), 341–370.

Demers, A. J., Petersen, K., Spreitzer, M. J., Terry, D. B., Theimer, M. M. & Welch, B. B. (1994), The Bayou architecture: Support for data sharing among mobile users, *in* 'IEEE Workshop on Mobile Computing Systems and Applications', Santa Cruz, California, pp. 2–7.

Dey, A., Abowd, G. D. & Salber, D. (2001), 'A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications', *Human Computer Interaction* pp. 97–167.

Douglis, F., Kaashoek, M. F., Li, K., Cáeres, R., Marsh, B. & Tauber, J. (1994), Storage alternatives for mobile computers, *in* 'First Symposium on Operating Systems Design and Implementation', Monterey, Californie, US, pp. 25–37.

Dourish, P. (2004), 'What we talk about when we talk about context', *Personal and Ubiquitous Computing* **8**(1), 19–30.

Dunham, M. H. & Helal, A. (1995), 'Mobile computing and databases: Anything new?', *SIGMOD Record* **24**(4), 5–9.

Dunham, M. H., Helal, A. & Balakrishnan, S. (1997), 'A mobile transaction model that captures both the data and movement behavior', *Mobile Networks and Applications* **2**(2), 149–162.

Dunham, M. H. & Kumar, V. (1999), Impact of mobility on transaction management, *in* 'international workshop on data engineering for wireless and mobile access', Seattle, WA, USA, pp. 14–21.

Elmasri, R. & Navathe, S. B. (2000), *Fundamentals of Database Systems, 3rd edition*, Addison Wesley Longman, Inc.

Faloutsos, C. (1996), *Searching Multimedia Databases by Content*, Kluwer Acedemic Inc.

Fife, L. D. & Gruenwald, L. (2003), 'Research issues for data communication in mobile ad-hoc network database systems', *SIGMOD Record* **32**(2), 42–47.

Ganguly, S. & Alonso, R. (1993), Query optimization in mobile environments, *in* 'Fifth Workshop on Foundations of Models and Languages for Data and Objects', Germany, pp. 1–17.

Garcia-Molina, H. (1982), 'Elections in a distributed computing system', *IEEE Transactions on Computers* **31**(1), 48–59.

Gessert, G. H. (1990), 'Four valued logic for relational database systems', *SIGMOD Record* **19**(1), 29–35.

Golfarelli, M., Maio, D. & Rizzi, S. (1998), Conceptual design of data warehouses from E/R schemes, *in* 'Hawaii International Conference on System Sciences', Kona, Hawaii.

Graefe, G. & Shapiro, L. D. (1991), Data compression and database performance, *in* 'ACM/IEEE-CS Symposium on Applied Computing', Kansas City, MO.

Gray, J., Helland, P., O'Niel, P. & Shasha, D. (1996), The dangers of replication and a solution, *in* 'ACM SIGMOD International Conference on the Management of Data', pp. 173–182.

Guttman, A. (1984), R-trees: A dynamic index structure for spatial searching, *in* 'ACM SIGMOD International Conference on the Management of Data', Boston, pp. 47–57.

Halevy, A. Y. (2001), 'Answering queries using views: A survey', *VLDB Journal* **10**(4), 270–294.

Han, J. & Fu, Y. (1994), Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases, *in* 'AAAI'94 Workshop on Knowledge Discovery in Databases', ACM Press, Seattle, WA, USA, pp. 157–168.

Harrison, B., Fishkin, K., Gujar, A., Mochon, C. & Want, R. (1998), Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces, *in* 'CHI '98'.

Hawick, K. & James, H. A. (2003), Middleware for context sensitive mobile applications, *in* 'Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing', Adelaide, Australia.

Heuer, A. & Lubinski, A. (1996), Database access in mobile environments, *in* 'Database and Expert Systems Applications', pp. 544–553.

Heuer, A. & Lubinski, A. (1998), Data reduction - an adaptation technique for mobile environments, *in* 'In Interactive Applications of mobile Computing (IMC'98)'.

Holliday, J., Agrawal, D. & Abbadi, A. E. (2000), Planned disconnections for mobile databases, *in* 'DEXA Workshop', pp. 165–172.

Huang, Y., Sistla, P. & Wolfson, O. (1994), Data replication for mobile computers, *in* 'ACM SIGMOD International Conference on the Management of Data', pp. 13–24.

Imielinski, T. & Badrinath, B. R. (1992), Querying in highly mobile distributed environments, *in* '18th International Conference on Very Large Data Bases', Vancouver, British Columbia, Canada.

Imielinski, T. & Badrinath, B. R. (1994), 'Mobile wireless computing: Challenges in data management', *Communications of the ACM* **37**(10), 18–28.

Imieliński, T. & Lipski, W. (1984), 'Incomplete information in relational databases', *JACM* **31**(4), 761–791.

Jing, J., Bukhres, O. & Elmagarmid, A. (1995), Distributed lock management for mobile transactions, *in* '15th International Conference on Distributed Computing Systems', Vancouver, Canada, pp. 118–125.

Jones, G. J. & Brown, P. J. (2004), Context-aware retrieval for ubiquitous computing environment, *in* 'Mobile and Ubiquitous Info Access Ws 2003', pp. 227–243.

Kaufman, L. & Rousseeuw, P. (1990), *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley and Sons, New York.

Keunning, G. H. & Popek, G. J. (1997), Automated hoarding for mobile computers, *in* '16th ACM Symposium on Operating Systems Principles', ACM Press, Saint Malo, France, pp. 264–275.

Kim, W. & Seo, J. (1991), 'Classifying schematic and data heterogeneity in multidatabase systems', *Computer* **24**(12), 12–18.

Korn, F., Jagadish, H. & Faloutsos, C. (1997), Efficiently supporting ad hoc queries in large datasets of time sequences, *in* J. Peckham, ed., 'ACM SIGMOD International Conference on the Management of Data', ACM Press, Tucson, AZ, pp. 289–300.

Lacroix, M. & Pirotte, A. (1976), 'Generalized joins', *SIGMOD Record* **8**(3), 14–15.

Ladin, R., Liskov, B., Shrira, L. & Ghemawat, S. (1992), 'Providing high availability using lazy replication', *ACM Transactions on Computer Systems* **10**(4), 360–391.

Larkby-Lahet, J., Santhanakrishnan, G., Amer, A. & Chrysanthis, P. K. (2005), 'Step: Self-tuning energy-safe predictors', *Mobile Data Management* pp. 125–133.

Lauzac, S. W. & Chrysanthis, P. K. (1998*a*), Programming views for mobile database clients, *in* 'DEXA Workshop', pp. 408–413.

Lauzac, S. W. & Chrysanthis, P. K. (1998*b*), Utilizing versions of views within a mobile environment, *in* 'the 9th ICCI'.

Lauzac, S. W. & Chrysanthis, P. K. (2002), View propagation and inconsistency detection for cooperative mobile agents, *in* 'Tenth International Conference on Cooperative Informaion Systems', pp. 107–124.

Law, C. & Siu, K.-Y. (2001), A bluetooth scatternet formation algorithm, *in* 'IEEE Symposium on Ad Hoc Wireless Networks'.

Lelewer, D. A. & Hirschberg, D. S. (1987), 'Data compression', *ACM Computing Surveys* **19**(3), 261–296.

Leopold, R. J., Miller, A. & Grubb, J. L. (1993), 'Iridium system: A new paradigm in personal communications', *Applied Microwave and Wireless* **5**(4), 68–78.

Lubinski, A. (2000), Small database answers for small mobile resources, *in* 'International Conference on Intelligent Interactive Assistance and Mobile Multimedia Computing (IMC2000), Rostock, November, 9-10'.

Lubinski, A. & Heuer, A. (2000), Configured replication for mobile applications, *in* 'Baltic DB&IS'2000', Vilnius, Lithuania.

Madria, S. K. (1998), Transaction models for mobile computing, *in* 'International Database Engineering and Application Symposium', Singapore, pp. 92–102.

Madria, S. K., Baseer, M. & Bhowmick, S. S. (2002), A multi-version transaction model to improve data availability in mobile computing, *in* 'CoopIS/DOA/ODBASE 2002', pp. 322–338.

Madria, S. K. & Bhargava, B. (1999), 'A transaction model to improve data availability in mobile computing', *Distributed and Parallel Databases* .

Madria, S., Mohania, M. & Roddick, J. F. (1998), A query processing model for mobile computing using concept hierarchies and summary databases, *in* K. Tanaka & S. Ghandeharizadeh, eds, 'Foundations of Database Organisation, FODO'98', Kobe, Japan, pp. 147–157. Republished in ¡i¿Information Organization and Databases¡/i¿. Boston, Kluwer Academic.

Maguire, G., Smith, M. & Beadle, H. (1998), Smartbadges: a wearable computer and communication system, *in* 'Codes/CASHE '98', Seattle, Washington, U.S.A.

Maier, D. (1983), *The theory of relational databases*, Computer Science Press, Rockville.

Mazumdar, S., Pietrzvk, M. & Chrysanthis, P. K. (2001), Caching constrained mobile data, *in* 'Proceedings of the 10th ACM International Conference on Information and Knowledge Management', pp. 442–449.

Moran, T. P. & Dourish, P. (2001), 'Introduction to special issue on context-aware computing', *Human Computer Interaction* **16**(2-3), 87–96.

Motro, A. (1988), ‘Vague: a user interface to relational databases that permits vague queries’, *ACM Trans. Inf. Syst.* **6**(3), 187–214.

Muthuraj, J., Chakravarthy, S., Varadarajan, R. & Navathe, S. B. (1993), A formal approach to the vertical partitioning problem in distributed database design, *in* ‘2nd International Conference on Parallel and Distributed Information Systems’, San Diego, California.

Nardi, B., ed. (1996), *Context and consciousness: Activity Theory and Human-Computer Interaction*, MIT Press.

Navathe, S. B., Ceri, S., Wiederhold, G. & Dou, J. (1984), ‘Vertical partitioning algorithms for database design’, *ACM Transactions on Database Systems* **9**(4), 680–710.

Navathe, S. B., Karlapalem, K. & Ra, M. (1996), ‘A mixed fragmentation methodology for initial distributed database design’, *Journal of Computer and Software Engineering* **3**(4).

Navathe, S. B. & Ra, M. (1989), Vertical partitioning for database design: A graphical algorithm, *in* ‘ACM SIGMOD International Conference on the Management of Data’, Vol. 18, Portland, pp. 440–450.

Neumann, K. & Maskarinec, M. (1997), Mobile computing within a distributed deductive database, *in* ‘1997 ACM symposium on Applied computing’, ACM Press, San Jose, California, United States, pp. 318–322.

Nicola, M. & Jarke, M. (2000), ‘Performance modeling of distributed replicated databases’, *IEEE Transactions on Knowledge and Data Engineering* **12**(4), 645–672.

Padovitz, A., Loke, S. W., Zaslavsky, A. B., Burg, B. & Bartolini, C. (2005), An approach to data fusion for context awareness, *in* ‘5th International and Interdisciplinary Conference CONTEXT’, Springer-Verlag, Paris, France.

Papadimitriou, C. (1986), *The theory of database concurrency control*, Computer Science Press, Inc., New York, NY, USA.

Phatak, S. H. & Badrinath, B. R. (1999), Data partitioning for disconnected client server databases, *in* ‘MobiDE’, pp. 102–109.

Pitoura, E. (1996), A replication schema to support weak connectivity in mobile information systems, *in* '7th International Conference on Database and Expert Systems Applications (DEXA)'.

Pitoura, E. & Bhargava, B. (1993), Dealing with mobility: Issues and research challenges, Technical report, Department of Computer Science, Purdue University, USA.

Pitoura, E. & Bhargava, B. (1994), Revising transaction concepts for mobile computing, *in* 'First IEEE Workshop on Mobile Computing Systems and Applications', Santa Cruz, CA, US, pp. 164–168.

Pitoura, E. & Bhargava, B. (1995*a*), 'A framework for providing consistent and recoverable agent-based access to heterogeneous mobile database', *SIGMOD Record* **24**(3), 44–49.

Pitoura, E. & Bhargava, B. (1995*b*), Maintaining consistency of data in mobile distributed environments, *in* '15th International Conference on Distributing Computing Systems', Vancouver, British Columbia, Canada, pp. 404–413.

Pitoura, E. & Bhargava, B. (1999), 'Data consistency in intermittently connected distributed systems', *IEEE Transactions on Knowledge and Data Engineering* **12**(5), 896–915.

Poosala, V., Ioannidis, Y. E., Haas, P. J. & Shekita, E. (1996), Improved histograms for selectivity estimation of range predicates, *in* 'ACM SIGMOD International Conference on the Management of Data', ACM Press, pp. 294–305.

Press, W., Teukolsky, S., Vetterline, W. & Flannery, B. (1996), *Numberical Recipes in C, The Art of Scientific Computing*, Cambridge University Press, Cambridge, MA.

Pu, C., Kaiser, G. E. & Hutchinson, N. (1988), Split-transactions for open-ended activities, *in* '14th Conference on Very Large Database', Los Altos, CA.

Ray, G., Haritsa, J. R. & Seshadri, S. (1995), Database compression: A performance enhancement tool, *in* 'International Conference on Management of Data', Pune, India.

Reiter, R. (1978), On closed world databases, *in* H. Gallaire & J. Minker, eds, 'Logic and Databases', Plenum Press, New York, pp. 55–76. Reprinted in

Artificial Intelligence and Databases. J. Mylopoulos and M.L. Brodie (eds.), Morgan Kaufmann, 248–258.

Reiter, R. (1986), 'A sound and sometimes complete query evaluation algorithm for relational databases with null values', *JACM* **33**(2), 349–370.

Rekimoto, J. (1996), Tilting operations for small screen interfaces, *in* 'ACM symposium on User interface software and technology', pp. 167–168.

Ren, Q. & Dunham, M. H. (2000), Using semantic caching to manage location dependent data in mobile computing, *in* 'Proceedings of the 6th annual international conference on Mobile computing and networking', Boston, Massachusetts, pp. 210–221.

Rice, S. P. & Roddick, J. F. (2000), Lattice-structured domains, imperfect data and inductive queries, *in* M. T. Ibrahim, J. Kng & N. Revell, eds, '11th International Conference on Database and Expert Systems Applications (DEXA2000)', Vol. 1873 of *LNCS*, Springer, Greenwich, London, UK, pp. 664–674.

Roddick, J. F. (1995), 'A survey of schema versioning issues for database systems', *Information and Software Technology* **37**(7), 383–393.

Roddick, J. F. (1997), The use of overcomplete logics in summary data management, *in* '8th Australasian Conference on Information Systems', Adelaide, Australia, pp. 288–298.

Roddick, J. F., Mohania, M. & Madria, S. (1999), Methods and interpretation of database summarisation, *in* T. Bench-Capon, G. Soda & A. Tjoa, eds, '10th International Conference on Database and Expert Systems Applications (DEXA'99)', Vol. 1677 of *LNCS*, Springer, Florence, Italy, pp. 604–615.

Roddick, J. F. & Patrick, J. D. (1992), 'Temporal semantics in information systems - a survey', *Information Systems* **17**(3), 249–267.

Roth, M. A. & Van Horn, S. J. (1993), 'Database compression', *SIGMOD Record* **22**(3), 31–39.

Roth, M., Korth, H. & Silberschatz, A. (1989), 'Null values in nested relational databases', *Acta Informatica* **26**(7), 615–642.

Sarndal, C., Swensson, B. & Wretman, J. (1992), *Model Assisted Survey Sampling*, Springer-Verlag, New York.

Schilit, B., Adams, N. & Want, R. (1994), Context-aware computing applications, *in* 'Workshop on Mobile Computing Systems and Applications', IEEE Computer Society, Santa Cruz, CA.

Schmidt, A. (2000), 'Implicit human computer interaction through context', *Personal Technologies* **4**(2).

Severance, D. G. (1983), 'A practitioner's guide to data base compression', *Information Systems* **8**(1), 51–62.

Shepherd, J., Harangsri, B., Chen, H. L. & Ngu, A. (1995), A two-phase approach to data allocation in distributed databases, *in* 'Database Systems for Advanced Applications', pp. 380–387.

Sheth, A. & Rusinkiewicz, M. (1990), Management of interdependent data: Specifying dependency and consistency requirements, *in* 'Workshop on the Management of Replicated Data', Houston, Texas, pp. 133–136.

Smith, M. (1999), Sensors in wearables, *in* 'International Conference on Wearable Computing', Stuttgart, Germany.

Snodgrass, R. (1987), 'The temporal query language TQUEL', *ACM Transactions on Database Systems* **12**(2), 247298.

Stanoi, I., Agrawal, D., El Abbadi, A., Phatak, S. H. & Badrinath, B. R. (1999), Data warehousing alternatives for mobile environments, *in* 'MobiDE', pp. 110–115.

Sudman, S. (1976), *Applied Sampling*, Academic Press, New York.

Tait, C. D., Lei, H., Acharya, S. & Chang, H. (1995), Intelligent file hoarding for mobile computers, *in* 'Mobile Computing and Networking', pp. 119–125.

Takagi, H. (1991), *Queueing Analysis, Vol. 1: Vacation and Priority Systems*, North-Holland.

Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J. & Hauser, C. H. (1995), Managing update conflicts in Bayou, a weakly connected replicated storage system, *in* '15th ACM Symposium on Operating Systems Principles', ACM, Copper Mountain Resort, Colorado.

Vassiliou, Y. (1979), Null values in data base management a denotational semantics approach, *in* 'SIGMOD '79: Proceedings of the 1979 ACM SIGMOD international conference on Management of data', ACM Press, New York, NY, USA, pp. 162–169.

Walborn, G. D. & Chrysanthis, P. K. (1995), Supporting semantics-based transaction processing in mobile database applications, *in* '14th IEEE Symposium on Reliable Distributed Systems'.

Walborn, G. D. & Chrysanthis, P. K. (1997), 'Pro-motion: Support for mobile database access', *Personal Technologies* **1**(3).

Walborn, G. D. & Chrysanthis, P. K. (1999), Transaction processing in pro-motion, *in* '1999 ACM Symposium on Applied Computing, SAC '99', San Antonio, TX, USA, pp. 389–398.

Want, R., Hopper, A., Falcao, V. & Gibbons, J. (1992), 'The active badge location system', *ACM Transactions on Information Systems* **10**(1), 91–102.

Ward, A., Jones, A. & Hopper, A. (1997), 'A new location technique for the active office', *IEEE Personal Communications* **4**(5), 42–47.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. & Alonso, G. (2000*a*), Database replication techniques: a three parameter classification, *in* '19th IEEE Symposium on Reliable Distributed Systems', IEEE Computer Society, Nürenberg, Germany.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. & Alonso, G. (2000*b*), Understanding replication in databases and distributed systems, *in* '20th International Conference on Distributed Computing Systems ICDCS'2000', IEEE Computer Society Technical Commitee on Distributed Processing, Taipei, Taiwan, R.O.C., pp. 264–274.

Wolfson, O. & Jajodia, S. (1995), 'An algorithm for dynamic data distribution in distributed systems', *Information Processing Letters* **53**, 113–119.

Wonnacott, R. J. & Wonnacott, T. H. (1985), *Introductory Statistics*, John Wiley, New York.

Yeo, L. H. & Zaslavsky, A. B. (1994), Submission of transactions from mobile workstations in a cooperative multidatabase processing environment, *in* '14th IEEE International Conference on Distributed Computing', pp. 372–379.

Yu, P. S., Dias, D. M. & Lavenberg, S. S. (1993), ‘On the analytical modeling of database concurrency control’, *Journal of the ACM* **40**(4), 831–872.

Yue, K.-b. (1991), ‘A more general model for handling missing information in relational databases using a 3-valued logic’, *SIGMOD Record* **20**(3), 43–49.

Zaniolo, C. (1982), Database relations with null values, *in* ‘PODS ’82: Proceedings of the 1st ACM SIGACT-SIGMOD on Principles of database systems’, ACM Press, New York, NY, USA, pp. 27–33.

Zaniolo, C. (1984), ‘Database relations with null values’, *Journal of Computer and System Sciences* **28**(1), 142–166.

Zaslavsky, A. B. (2004), Mobile Agents: Can They Assist with Context Awareness?, *in* ‘Proceedings of the 2004 IEEE International Conference on Mobile Data Management’.

Zaslavsky, A. B., Chrysanthis, P. K. & Kumar, V. (2004), ‘Mobility in databases and distributed systems: Summing up achievements of the past decade’, *Mobile Networks and Applications* **9**, 455–457.

Zaslavsky, A. B. & Tari, Z. (1998), ‘Mobile computing: Overview and current status’, *Australian Computer Journal* **30**(2), 42–52.

Zenel, B. & Duchamp, D. (1995), Intelligent communication filtering for limited bandwidth environments, *in* ‘IEE Fifth Workshop on Hot Topics in Operating Systems’, Rosario WA.