CURRENT ISSUES

In this chapter, we discuss four topics that are of growing importance in distributed database management. The topics are data warehousing (Section 16.2), the World Wide Web (WWW, or Web for short) and its effect on distributed data management (Section 16.3), push-based technologies for data dissemination (Section 16.4), and mobile database management (Section 16.5). In this introduction, we will justify the choice of these topics before going into the details.

In previous chapters, we pointed several times to the changes in the technologies underlying distributed DBMSs. One of the changes we highlighted was the emergence of broadband networks, and we discussed the effects of these networks on the architecture and query processing/optimization routines of distributed DBMSs. Another major change is the expanding use and reach of the Internet, the proliferation of *intranets* and the ongoing advances in the World Wide Web. This has fueled the development of a wide range of data intensive applications and information dissemination systems. In such an environment, providing integrated access to multiple, distributed databases and other heterogeneous data sources has more than ever been the focus of the database community. One of these data-intensive applications is decision support systems. Over the years, companies have built *operational* databases to support their day-to-day operations with On-Line Transaction Processing (OLTP) applications. OLTP applications, such as airline reservation or banking systems, are transaction-oriented and update-intensive. They need extensive data control and availability, high multiuser throughput and predictable, fast response times. The users are clerical. Operational databases are medium to large (up to several gigabytes). In effect, distributed databases have been used to provide integrated access to multiple operational databases.

Decision support applications have been termed *On-line Analytical Processing* (OLAP) [Codd, 1995] to better reflect their different requirements. OLAP applications, such as trend analysis or forecasting, need to analyze historical, summarized data coming from operational databases. They use complex queries over potentially very large tables and are read-intensive. Because of their strategic nature, response time is important. The users are managers or analysts. Performing OLAP queries directly over distributed operational databases raises two problems. First, it hurts the OLTP applications' performance by competing for local resources. Second, the overall response time of the OLAP queries can be very poor

because large quantities of data must be transferred over the network. Furthermore, most OLAP applications do not need the most current versions of the data, and thus do not need direct access to operational data. The now-popular solution to this problem is *data warehousing* [Inmon, 1992] which extracts and summarizes data from the operational databases in a separate database, dedicated to OLAP. Data warehousing is often considered an alternative to distributed databases, but we will see in Section 16.2 that these are complementary technologies.

With the ongoing advance in Web technology, everyone today can publish information on the Web independently at any time. The flexibility and autonomy of producing and sharing information on Web is impressive. On the other hand, one has to learn to deal with the rapid increase of volume and diversity of online information and the constant changes of information sources in number, content, and location. The diversity of applications that are deployed on the Web require different data delivery modes and protocols in different frequencies [Liu et al., 1998]. The design space of data delivery alternatives can be characterized in a number of ways. We examine these alternatives in some detail in Section 16.1. Two important points in this design space are *pull-based* systems, where clients access data servers to retrieve the required data, and *push-based* systems, where the servers "deliver" data to clients without waiting to be asked.

The typical method of accessing data on the Web is pull-based. The Web was initially designed to provide a distributed hypertext system on the Internet. It has had great and widespread success, due primarily to its graphical interface and its browsers—for example, Internet Explorer and Netscape, which make navigation through distributed documents easy. Since its inception the Web interface has been made increasingly multimedia and interactive. The Web has become a truly global network, now linking hundreds of thousands of servers. Organizations are extensively using the Web in intranets, which are private networks of Web servers, for various applications such as information publishing, workflow, groupware and accessing information systems.

Web technology remains document-oriented and provides information retrieval engines to search distributed documents. However, the database industry has developed gateways to interface Web documents with databases. As a result, the number of heterogeneous data sources that can be accessed from a Web browser is gigantic. Thus, providing integrated access to multiple, distributed data sources on the Web has become a major technical challenge. Distributed database and data warehouse technologies are useful here, but need to be significantly extended to deal with several issues. One is the very wide heterogeneity of the data sources and their computing capabilities, ranging from highly structured databases to unstructured files with semistructured documents as an interesting intermediate point. Another issue is the need to scale up to high numbers of distributed data sources that can be dynamically added or dropped. We consider the relationship between Web technology and distributed data management in Section 16.3.

Push-based technologies, on the other hand, are being proposed in response to communication asymmetry exhibited by many applications, such as news delivery, software distribution, and traffic information systems [Franklin and Zdonik, 1997]. Communication asymmetry refers to the fact that the communication from

the clients to the server is more restricted than the communication from the server to the clients. In these environments, it may make more sense to "push" data from the servers without waiting for the clients to "pull" them. There is significant commercial interest in push-based information delivery, including announcements by major Web browser vendors about plans to incorporate push into their products. We discuss the issues in designing these systems in Section 16.4.

In addition to the proliferation of broadband networks and the Internet, a third major technological change affecting distributed data management is the emergence of mobile networks. In this context, we are not referring to satellite-based systems, but to digital cellular networks. These networks have quite interesting characteristics which require reconsideration of major data management algorithms and protocols. We consider these issues in Section 16.5.

16.1 DATA DELIVERY ALTERNATIVES

We characterize the data delivery alternatives along three orthogonal dimensions: delivery modes, frequency and communication methods. The entire design space is depicted in Figure 16.1, which is a combination of characterizations proposed in [Franklin and Zdonik, 1997] and [Liu et al., 1998]. In the remainder, we discuss the design alternatives along the axes, but not every possible alternative system architecture.

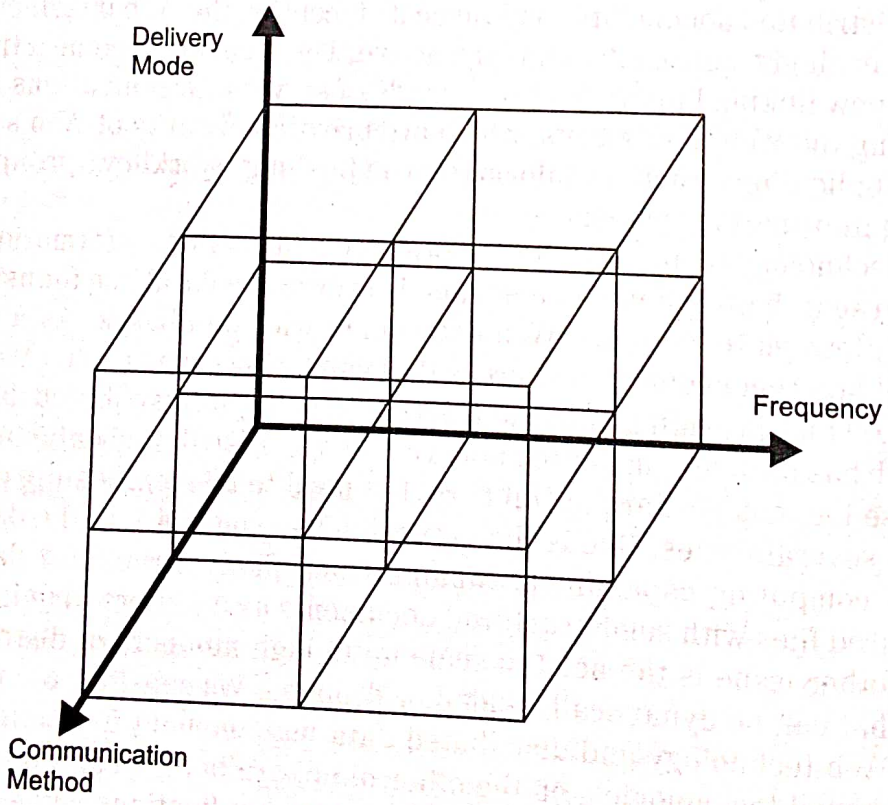

Figure 16.1. Data Delivery Alternatives

The alternative delivery modes are pull-only, push-only and hybrid. In the *pull-only* mode of data delivery, the transfer of data from servers to clients is initiated by a client pull. When a client request is received at a server, the server responds by locating the requested information. The main characteristic of pull-based delivery is that the arrival of new data items or updates to existing data items are carried out at a server without notification to clients unless clients explicitly poll the server. Also, in pull-based mode, servers must be interrupted continuously to deal with requests from clients. Furthermore, the information that clients can obtain from a server is limited to when and what clients know to ask for. Conventional DBMSs (including relational and object-oriented ones) offer primarily pull-based data delivery.

In the *push-only* mode of data delivery, the transfer of data from servers to clients is initiated by a server push in the absence of any specific request from clients. The main difficulty of the push-based approach is in deciding which data would be of common interest, and when to send them to clients (periodically, irregularly, or conditionally). Thus, the usefulness of server push depends heavily upon the accuracy of a server to predict the needs of clients. In push-based mode, servers disseminate information to either an unbounded set of clients (random broadcast) who can listen to a medium or selective set of clients (multicast), who belong to some categories of recipients that may receive the data.

The hybrid mode of data delivery combines the client-pull and server-push mechanisms. The continual query approach described in [Liu et al., 1996] presents one possible way of combining the pull and push modes: namely, the transfer of information from servers to clients is first initiated by a client pull, and the subsequent transfer of updated information to clients is initiated by a server push.

The hybrid mode represented by the continual queries approach can be seen as a specialization of push-only mode. The main difference between hybrid mode and push-only mode is the initiation of the first data delivery. More concretely, in a hybrid mode, clients receive the information that matches their profiles from servers continuously. In addition to new data items and updates, previously existing data that matches the profile of a client who initially pulled the server are delivered to the client immediately after the initial pull.

There are three typical frequency measurements that can be used to classify the regularity of data delivery. They are *periodic*, *conditional*, and *ad-hoc* or *irregular*. In periodic delivery, data are sent from the server to clients at regular intervals. The intervals can be defined by system default or by clients using their profiles. Both pull and push can be performed in periodic fashion. Periodic delivery is carried out on a regular and pre-specified repeating schedule. A client request for IBM's stock price every week is an example of a periodic pull. An example of periodic push is when an application can send out stock price listing on a regular basis, say every morning. Periodic push is particularly useful for situations in which clients might not be available at all times, or might be unable to react to what has been sent, such as in the mobile setting where clients can become disconnected.

In conditional delivery, data are sent from servers whenever certain conditions installed by clients in their profiles are satisfied. Such conditions can be as simple

as a given time span or as complicated as event-condition-action rules. Conditional delivery is mostly used in the hybrid or push-only delivery systems. Using conditional push, data are sent out according to a pre-specified condition, rather than any particular repeating schedule. An application that sends out stock prices only when they change is an example of conditional push. An application that sends out a balance statement only when the total balance is 5% below the pre-defined balance threshold is an example of hybrid conditional push. Conditional push assumes that changes are critical to the clients, and that clients are always listening and need to respond to what is being sent. Hybrid conditional push further assumes that missing some update information is not crucial to the clients.

Ad-hoc delivery is irregular and is performed mostly in a pure pull-based system. Data are pulled from servers to clients in an ad-hoc fashion whenever clients request it. In contrast, periodic pull arises when a client uses polling to obtain data from servers based on a regular period (schedule).

The third component of the design space of information delivery alternatives is the communication method. These methods determine the various ways in which servers and clients communicate for delivering information to clients. The alternatives are *unicast* and *one-to-many*. In unicast, the communication from a server to a client is one-to-one: the server sends data to one client using a particular delivery mode with some frequency. In one-to-many, as the name implies, the server sends data to a number of clients. Note that we are not referring here to a specific protocol; one-to-many communication may use a multicast or broadcast protocol.

We should note that this characterization is subject to considerable debate. It is not clear that every point in the design space is meaningful. Furthermore, specification of alternatives such as conditional and periodic (which may make sense) is difficult. However, it serves as a first-order characterization of the complexity of emerging distributed data management systems.

16.2 DATA WAREHOUSING

Data warehousing refers to a collection of technologies aimed at improving decision making. A data warehouse can be defined as a subject-oriented collection of data integrated from various operational databases. Information is classified by subjects of interest to business analysts, such as customers, products and accounts. Data warehouses are reminiscent of older mainframe-based reporting systems. However, they are based on open systems and relational databases. Furthermore, a data warehouse can be directly accessed by end-users on powerful workstations via sophisticated, graphical analysis tools, thus eliminating the need to rely on skilled application programmers.

Warehouse information is historical in nature, reflecting OLTP transactions that have occurred in the previous months or years. Thus, to facilitate access and analysis, warehouse data are typically summarized and aggregated in a multidimensional fashion. OLAP operations manipulate the data along the multiple dimensions, for instance to increase or decrease the level of aggregation. Warehouse data are generally read-only.

In the rest of this section, we present the various architectures for data warehousing, the OLAP multidimensional data model and the OLAP servers. We conclude with open issues.

16.2.1 Architectures

Figure 16.2 illustrates a simplified architecture of a data warehouse, with its various elements. One or more source databases, containing operational data updated by OLTP applications, are integrated in a single target database, or data warehouse. The target database is accessed through queries by desktop applications such as query and analysis, reporting and data mining tools. Popular desktop applications for data analysis are spreadsheet programs.

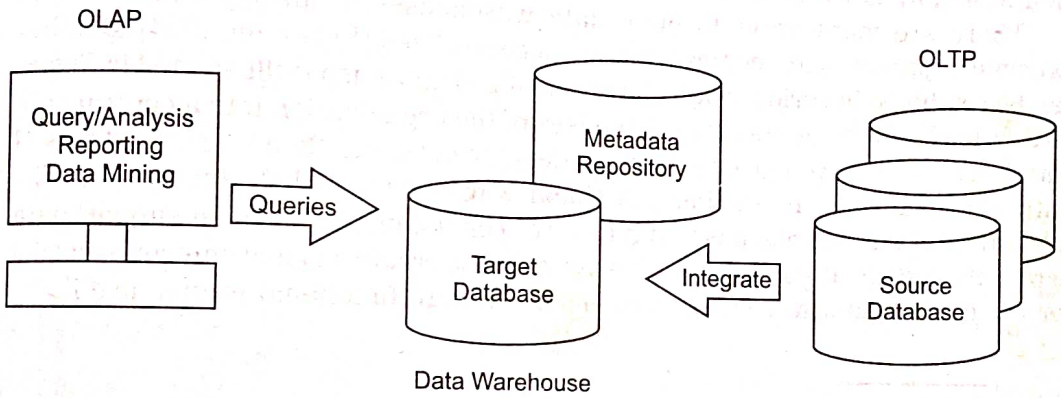

Figure 16.2. Architecture of a Data Warehouse

The metadata repository is a separate database that keeps track of the data currently stored in the data warehouse. Typical metadata include descriptions of target tables with their source definitions. The metadata repository is useful to isolate the data warehouse from changes in the schema of source databases. For instance, when a change occurs in a source database schema, the data warehouse administrator can simply update the repository, and the change is automatically propagated to the target database and the OLAP applications.

The integration of multiple source databases in a data warehouse raises several issues. At design time, an integrated schema must be defined. Schema integration in a data warehouse is similar to schema integration in multidatabase systems (see Chapter 15), and must deal with semantic conflicts across distinct, autonomous source databases. Schema integration in data warehousing typically assumes that each source database provides a relational interface. Thus schema integration can be done by defining relational views over the source tables [Roussopoulos, 1998].

Populating the data warehouse relies on data extraction, cleaning and loading utilities [Chaudhuri and Dayal, 1997]. Data extraction is implemented using gateways supporting standard interfaces, such as Open Database Connectivity (ODBC).

Data are also usually cleaned to reduce inconsistencies, like missing entries or invalid values. After extraction and cleaning, data are loaded in the data warehouse, with additional processing like summarization and aggregation. Loading data essentially means materializing the relational views of the integrated schema. Furthermore, data in the data warehouse are organized for efficient query processing with various kinds of indices.

After it is populated, the data warehouse must be refreshed from time to time to reflect updates to the source databases. Refreshing is usually done periodically, for instance daily or weekly. It can also be done immediately after every update for OLAP applications that need to see current data. To avoid populating entire tables, only updates to the source data should be propagated to the data warehouse. This is done using asynchronous replication techniques that perform incremental maintenance of replicas from primary copies.

There are many ways to build data warehouses in an organization. Two extreme approaches are centralized and decentralized [Eckerson, 1994]. But they can be combined in various ways. The centralized approach is illustrated in Figure 16.3. It loads all the operational databases of the organization into a central data warehouse. Functional groups, such as departments, can then extract subsets of data and load them in smaller functional warehouses called *datamarts*. The advantage of this approach is that it ensures consistency of decision-support data across the organization. However, it is very hard to create a global enterprise model for the central data warehouse and ensure that all functional groups use it.

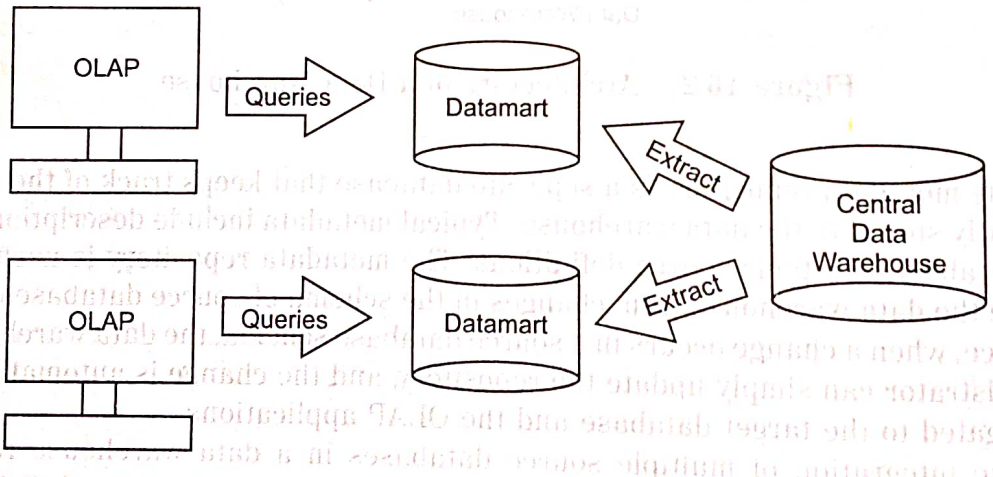

Figure 16.3. Centralized Data Warehouse

The decentralized approach is illustrated in Figure 16.4. The data warehouse here is virtual and provides the OLAP applications with a global view of the data, using a global data dictionary managed by a data warehouse gateway. Global queries are decomposed into local queries sent to the operational DBMSs, and the results are integrated by the data warehouse gateway. This is a direct application of distributed database technology with one global schema [Hull and Zhou, 1996].

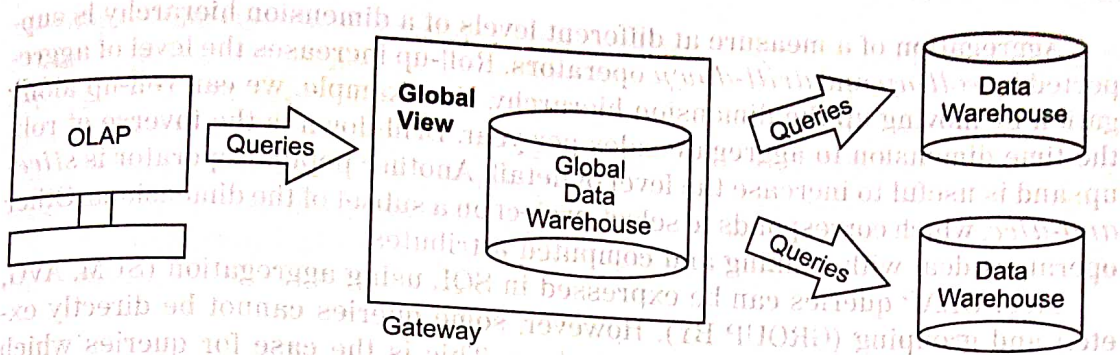

Figure 16.4. Decentralized Data Warehouse

16.2.2 OLAP Data Model

The OLAP data model is *multidimensional*. The data is represented by a multidimensional array of *numeric measures*, such as sales or revenue, which is useful for analysis. The dimensions uniquely determine the measure. In a sales data warehouse, for instance, interesting dimensions may include time of sale, location and product (Figure 16.5). Each dimension can be represented by one or more attributes. For instance, the time dimension which is of particular interest to OLAP applications can have attributes date, week, month, quarter and year. The attributes of a dimension are often organized as a hierarchy like year → quarter → month → week or industry → category → product.

OLAP operations enable end users, such as business analysts, to manipulate multidimensional data with a powerful spreadsheet style interface. Most of these operations deal with aggregation. Aggregation of a measure along a subset of the dimensions is done by *pivoting* (reorienting) the multidimensional view of the data. For example, let us consider the three-dimensional array in Figure 16.5. Pivoting that array along the location and time dimensions would produce a two-dimensional array in which each point (x, y) gives the aggregated sales for location x at time y.

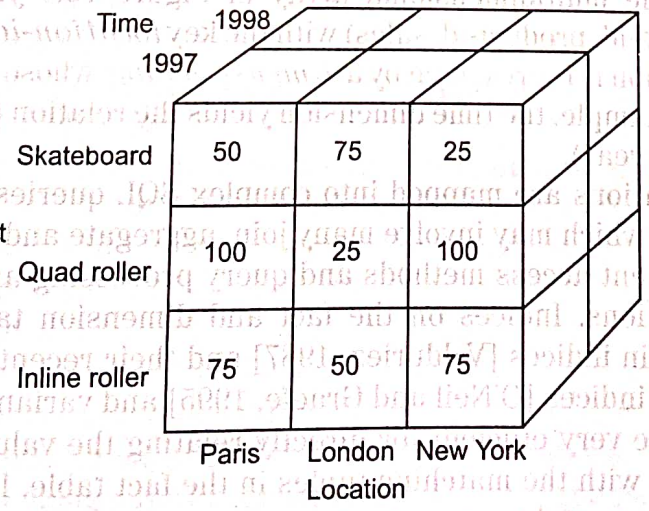

Figure 16.5. Multidimensional Data

Aggregation of a measure at different levels of a dimension hierarchy is supported by *roll-up* and *drill-down* operators. Roll-up increases the level of aggregation by moving up the dimension hierarchy. For example, we can roll-up along the time dimension to aggregate sales per year. Drill-down is the inverse of roll-up, and is useful to increase the level of detail. Another popular operator is *slice-and-dice*, which corresponds to select-project on a subset of the dimensions. Other operators deal with ranking and computed attributes.

Most OLAP queries can be expressed in SQL using aggregation (SUM, AVG, etc.) and grouping (GROUP BY). However, some queries cannot be directly expressed and require significant extensions. This is the case for queries which involve ranking (e.g., select the top ten cities ranked by sales).

16.2.3 OLAP Servers

There are two main approaches to implementing data warehouse servers [Colliat, 1996]. *Multidimensional OLAP* (MOLAP) servers directly support OLAP operations on multidimensional data structures, whereas *Relational OLAP* (ROLAP) servers extend relational databases to support OLAP operations.

MOLAP servers store the data in a multidimensional data structure that is dedicated to OLAP operations. The data structure typically has two-levels to deal with large data sets that do not fit in main memory. With such a data structure, OLAP operations can be processed very efficiently in memory [Finkelstein, 1995]. However, changing the multidimensional array, e.g., by adding another dimension, requires a complete reconstruction of the array. Furthermore, MOLAP servers do not scale up to very large databases because the core index must fit in memory. A final problem is that there is no standard multidimensional data model, and MOLAP servers are substantially different in addressing the requirements of OLAP.

ROLAP servers are extended relational DBMS or intermediate servers in front of relational DBMS that store the data in relations. A multidimensional array is represented by two kinds of relations. The *fact table* relates the dimensions to the measures with a relation whose primary key is the set of dimension identifiers. For example, the multidimensional array in Figure 16.5 yields the fact table (location-id, time-id, product-id, sales) with the key *location-id, time-id, product-id*. Each dimension is represented by a *dimension table* whose key is the dimension identifier. For example, the time dimension yields the relation (time-id, date, week, month, quarter, year).

OLAP operations are mapped into complex SQL queries on the fact and dimension tables, which may involve many join, aggregate and group-by operators. Therefore, efficient access methods and query processing are necessary to deal with large relations. Indices on the fact and dimension tables are important. Furthermore, join indices [Valduriez, 1987] and their recent extensions, such as bit-mapped join indices [O'Neil and Graefe, 1995] and variant indices [O'Neil and Quass, 1997] are very efficient for directly relating the values of attributes in a dimension table with the matching tuples in the fact table. Like indices, *materialized views* are useful to precompute summary data and aggregates. The major issues are the efficient updating of materialized views using incremental

techniques and the transformation of user queries in terms of materialized views [Blakeley et al., 1986]. Another complementary solution for efficient OLAP processing is parallelism (see Chapter 13). In fact, a major application of parallel database systems is data warehousing. Compared to MOLAP servers, ROLAP servers have many advantages, such as a standard data model, and scale up to very large databases. However, the mapping of OLAP queries to SQL is hard to make efficient. For small to medium-size data, warehouses, MOLAP servers are likely to be more efficient.

16.2.4 Research Issues

Because data warehousing has been a fast growing market for software products and services, the practice has been and still is preceding research. All the leading software vendors, in particular relational DBMS vendors, are offering data warehouse systems. However, most of the current products present severe limitations in terms of flexibility, efficiency and scalability. To overcome these limitations, important research issues must be addressed.

Data quality is becoming a major issue as more and more strategic decisions rely on the data warehouse. Typical data quality parameters are data accessibility, interpretability, usefulness, believability and validation. To improve data quality, data warehouse design and evolution should incorporate formal models of information quality. The DWQ project in Europe [Sellis et al., 1997] addresses this proble by developing a "semantic foundation to allow the designers of data warehouse to link the choice of deeper models, richer data structures and rigorous implementation techniques to quality-of-service factors in a systematic manner." It does so by developing advanced data warehouse components with a reasoning capability based on description logic.

Data warehouse management is also getting difficult with the deployment of decentralized datamarts as an alternative to the centralized data warehouse approach. For operational reasons, these datamarts are fairly autonomous, and tend to grow and duplicate information without global consistency control within the organization. Besides the data quality issues mentioned above, architectural issues should be addressed to avoid uncontrolled data duplication and yield flexibility and scalability. This requires a better combination of data warehouse and distributed database technologies.

Materialized view maintenance in a data warehouse is significantly more difficult than in DBMS [Widom, 1995], [Quass and Widom, 1997]. Views stored in the data warehouse tend to be more complicated than conventional views, with aggregated and summarized information [Dar et al., 1996a], [Gupta et al., 1995], [Griffin and Libkin, 1995]. Thus, it is not always possible to express view definitions in SQL. Furthermore, views may be based on histories of source data and thus should include temporal capabilities. Finally, there might be multiple views of the same data—for instance, to support different kinds of analysis. Therefore, more research is needed to design view languages for data warehouses and to revisit view maintenance algorithms accordingly.

The problems of physical design and query optimization must also be revisited. New kinds of indices, buffer management strategies, cost models and query transformations should be devised by considering the special requirements of data warehouses, in particular, read-only queries with aggregation.

Data integration, including data extraction, cleaning, loading and refreshing, still presents serious challenges. In addition to schema inconsistencies traditionally studied in heterogeneous data integration, data cleaning should emphasize data inconsistencies. Another problem is *change detection*, which detects and propagates the changes in the source data to the data warehouse. Data sources can be classified according to their ability for change detection [Widom, 1995]. *Cooperative sources* provide trigger capabilities which ease the programming of automatic notifications of changes. *Logged sources* maintain a log from which changes can be extracted. *Queryable sources* can be queried by the data warehouse. Finally, *snapshot sources* can only be copied off-line. The WHIPS data warehousing project at Stanford [Hammer et al., 1995a], [Labio et al., 1997] addresses the problem of automatic change detection and incremental data integration in the data warehouse.

Another problem related to data integration is improving the freshness of warehouse data. Replication techniques have been used successfully for refreshing the data warehouse periodically [Helal et al., 1997]. But for some applications, such as on-line financial analysis, very high freshness is crucial and traditional replication solutions do not work [Gray et al. 1996]. Asynchronous replication techniques that support near real-time constraints [Pacitti et al., 1998] are necessary here.

16.3 WORLD WIDE WEB

The world-wide, frontier-free dimension of the Web, combined with its portability and ease of use, makes it a major enabling element of the future information society. The Web has grown exponentially and the number and diversity of users (individuals, enterprises, governments, etc.) and applications (education, on-line publication, electronic commerce, etc.) are growing continuously. It has become something that no organization, public or private, which expects to develop and to grow can afford to ignore.

The development has been so rapid that many technical problems have become acute, in particular, security and information access. Searching for relevant information is becoming increasingly difficult; existing tools, for example, search engines such as Alta Vista and Yahoo, support primitive searching of documents and cannot exploit data structures. Nor can they be used to provide fast access to many data sources of data, in particular, within intranets.

Therefore, the Web provides many avenues of research in distributed databases[1] which we will discuss. In the rest of this section, we briefly present the architecture of the Web and introduce its standard protocols. We then describe architectures that support database access and information integration on the

[1]Some researchers like to see the Web as a gigantic distributed database.

Web. We also define semistructured data which are proliferating on the Web. We end with short presentations of research projects and open issues.

16.3.1 Architecture and Protocols

The Web architecture is client-server, with its simple standard communication protocol HTTP (HyperText Transfer Protocol) implemented on top of TCP/IP, the ubiquitous Internet protocol. With HTTP, any client browser can request a document on a Web server using its *uniform resource locator* (URL). The entry point to a Web site is its *home page*, identified by a URL. For example, the URL of the World Wide Web consortium (W3C), which is responsible for defining the Web standards, is http://www.w3c.org[2]. The home page refers to other documents' URLs, thus providing for hypertext navigation. HTTP is efficient in allowing high numbers of independent, stateless connections. The counterpart is that it does not maintain any context between client and server, which makes it hard to support sessions.

HTML (HyperText Markup Language) is the standard page description language for the Web. It allows users to describe the content of a page (text, sound, image, etc.) with markups for displaying. HTML should be used only to mark up information according to its meaning, regardless of how it will be displayed by the browser. Conceptually, HTML and HTTP are independent of platforms and protocols, thereby solving the problem of portability of client graphical interfaces. However, HTML is quite permissive, and this has been exploited by Web browsers to include proprietary markups for displaying. As a result, some HTML documents cannot be properly displayed by various browsers. To provide browser independence, an extended version of HTML called XML (extended Markup Language) has been defined by the W3C. XML is a subset of SGML (Standard Generalized Markup Language) and provides a clean separation between content structure and presentation. With XML, one can encode structure information in documents much more precisely than with HTML. Besides its exploitation for better displaying by Web browsers, XML-encoded information can also be used for searching in more sophisticated SQL-like ways. XML is already having a major impact as a standard for data access.

HTTP enables access to *static* Web pages, i.e., pages already stored in files in the Web server. However, Web applications that access a DBMS or some other file server must create *dynamic* Web pages. The CGI (Common Gateway Interface) protocol makes this possible by enabling a Web server to call an executable program specified in the URL. A typical example of this kind of program is one that performs SQL access. The output produced by the program is returned to the Web server and is used to construct the dynamic Web page.

16.3.2 Database Access

With the Web, the client is highly portable and is sometimes called "universal client". This yields a new form of client/server architecture called *three-tier client/*

[2]All information regarding Web standards is available at that site.

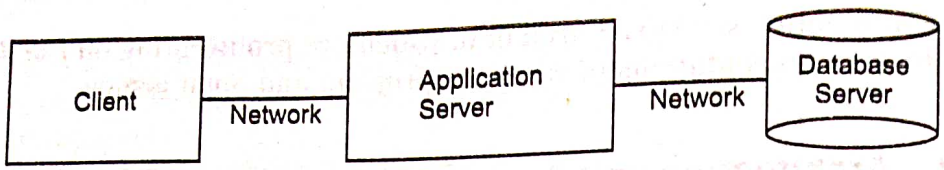

Figure 16.6. Three-Tier Client/Server Architecture

server (Figure 16.6). With traditional (two-tier) client/server architectures, the server runs the database programs, and the client runs the application programs and the graphical interface. The problems with this architecture are now well understood. As the number of application programs increases, the clients get "fatter" and require more storage and computing power. Furthermore, propagating changes to application programs on many heterogeneous clients may be difficult.

With the three-tier client/server architecture, an application server is introduced to run application programs. The client is a Web browser dedicated to the graphical interfaces. Application logic that may be needed at the client, e.g., to check data entry, can be supported through *applets*, which are small application programs downloaded from the application server. Applets are typically programmed in Java, a secure programming language, and run in a restricted address space (the "sandbox") which prevents disk access. As a side effect, the client does not need as much computing power as before. The concept of a *Network Computer* (NC), essentially a PC without a hard disk, has been proposed by Oracle as a cheaper alternative to the PC. It is very likely that PC and NC will complement each other in performing different tasks.

The three-tier architecture can naturally generalize to n-tier with various application servers. Figure 16.7 illustrates this generalization to enable database access from a Web server, as most DBMS do. The Web browser communicates with the Web server using HTTP. The database gateway is a CGI program that performs the mapping between HTML inputs and query strings, and between result tuples and HTML report forms. Nguyen and Srinivasan [1996] propose a general-purpose cross language variable substitution to achieve this in the DB2 WWW Connection system. The database gateway is called by the Web server using the URL and associated inputs (for the query), and sends the corresponding query to the database server. After the query has been executed, it transforms the result tuples into dynamic HTML pages. Thus, the application developer can use both HTML for creating query forms and reports, and the database language (SQL) for querying.

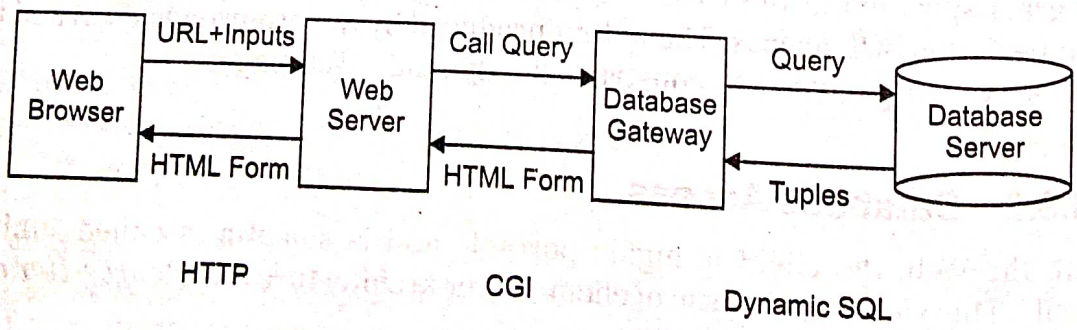

Figure 16.7. Database Access from a Web Browser

16.3.3 Semistructured Data

The Web is usually regarded as an interconnected collection of unstructured documents. However, a large number of structured data sources are now becoming available on the Web using database gateways. These sources include both free and commercial databases on product information, stock market information, real estate, automobiles, and entertainment. The interface to such sources is typically a collection of *fill-out forms*. The query answer usually takes the form of an HTML document that is very structured, and can be parsed and converted into a set of tuples or more complex data types. There are other structured information sources that are available online but not on the Web, such as name servers, bibliographic sources, and university- and company-wide information systems, and they too provide query interfaces.

Consequently, the data available on the Internet is structured in many different ways. The two extremes are fully unstructured data, such as raw text and image, and fully structured data, such as relational or object databases. But most data, such as HTML and SGML data, are somewhere in between, and have thus been termed *semistructured* [Abiteboul, 1997], [Suciu, 1997]. Semistructured data differ from structured data in many important ways. The structure may be irregular because of data heterogeneity, implicit as in SGML, or partial. The schema may be very large, rapidly evolving or completely ignored in queries for Information Retrieval (IR) or browsing.

Semistructured data are typically modeled as a labeled graph whose nodes are labeled structured objects or atomic values, and whose edges are references. Data in the labeled graph are self-describing and have no schema. Let us illustrate such labeled graphs with the well-known Object Exchange Model (OEM) [Papakonstantinou et al., 1995]. An OEM object consists of

1. a label which is the name of the object class
2. a type which is either atomic (integer, string, etc.) or set
3. a value which is either atomic or a set of objects
4. an optional object identifier

Figure 16.8 shows an OEM object with the label "Highlights 98", whose value is a set of the major sport events in 1998, like the Roland Garros tennis tournament in Paris and the soccer World Cup in France. As the example suggests, it is very easy to integrate heterogeneous data, since they are self-describing. However, type information is embedded in the objects and is difficult to extract. To describe the structure of semistructured objects, graph schemas [Buneman, 1997] and data guides [Goldman 1997] have been proposed.

Research on semistructured data is only beginning. Traditional database techniques rely heavily on the database schema, and are therefore not suited for semistructured data. New (or extensions of existing) database languages, techniques and architectures are necessary.

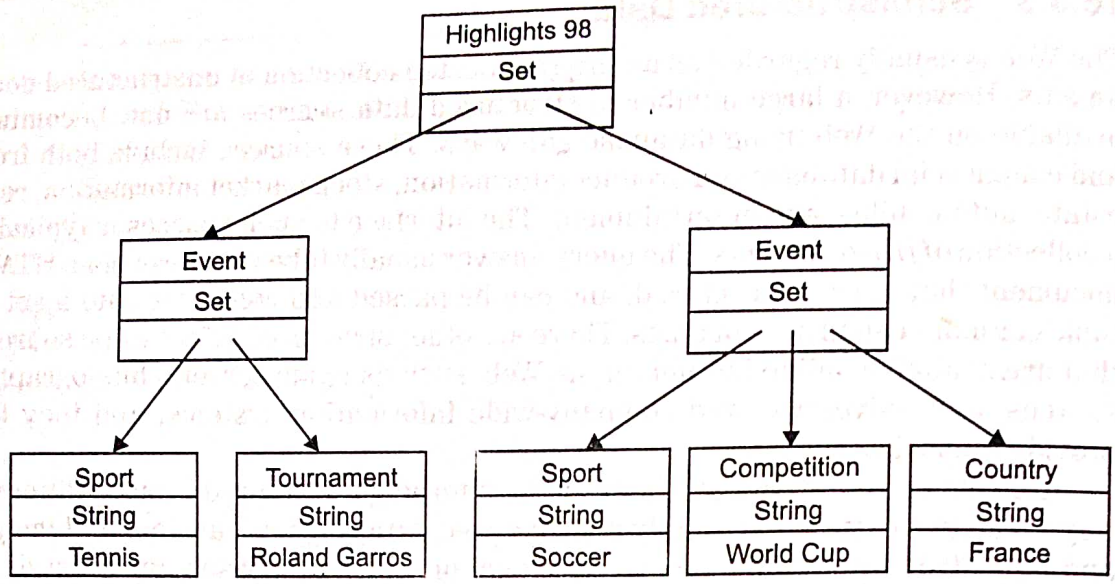

Figure 16.8. An OEM Object

16.3.4 Architectures for Information Integration

Integrating information from data sources over the Internet requires creating some form of integrated view to allow for distributed querying. The context of the Internet raises a number of issues for information integration that are far more difficult than those of multidatabase systems. First, the number of data sources may be very high, thereby making view integration and conflict resolution a problem. Second, the space of data sources is very dynamic, so adding or dropping a data source should be done with minimal impact on the integrated view. Third, the data sources may have different computing capabilities, ranging from full-featured DBMS to simple files. This is unlike multidatabase systems or data warehousing, which assume data sources with an SQL interface. Finally, data sources may be unstructured or semistructured, thus providing virtually no information for view integration.

To address these problems, the database research community has revisited the multidatabase architecture with data source *wrappers* and *mediators* (Figure 16.9).

- For each data source, a wrapper exports some information about its source schema, data, and query processing capabilities [Cluet et al., 1998].
- A mediator centralizes the information provided by the wrappers in a unified view of all available data (stored in the global data dictionary), decomposes the user query in smaller queries (executable by the wrappers), gathers the partial results and computes the answer to the user query.

There is no consensus on how wrappers describe their sources' capabilities, nor on how much of this information is exposed to the mediator. But the mediator-wrapper model itself is a widely adopted abstraction for the information integra-

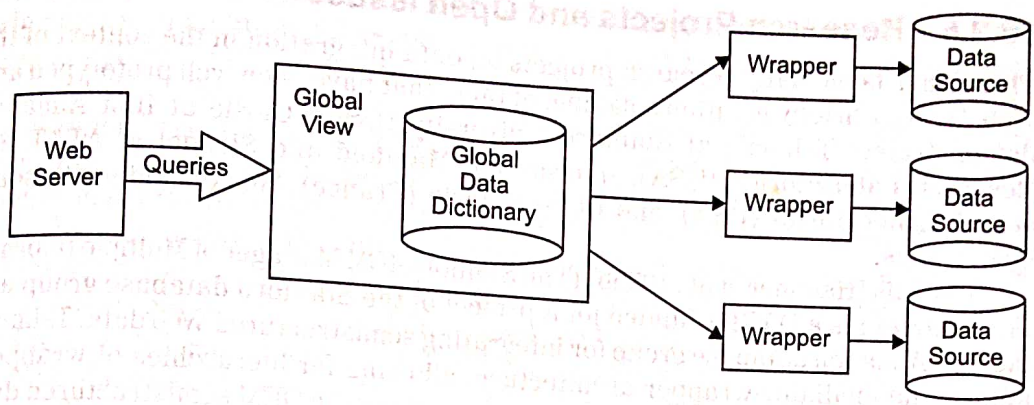

Figure 16.9. Mediator-Wrapper Architecture

tion problem. The mediator-wrapper architecture differs fundamentally from a data warehouse in that integrated data is not materialized. Thus, it is rather complementary, since a mediator can be used as a source database for a data warehouse.

The mediator-wrapper architecture has several advantages. First, the specialized components of the architecture allow the various concerns of different kinds of users to be handled separately. Second, mediators typically specialize in a related set of data sources with "similar" data, and thus export schemas and semantics related to a particular domain. The specialization of the components leads to a flexible and extensible distributed system. Figure 16.10 illustrates a hierarchy of specialized mediators, with one IR mediator for various search engines, one DB mediator for heterogeneous databases, and an IR/DB mediator that provides IR and database query capabilities.

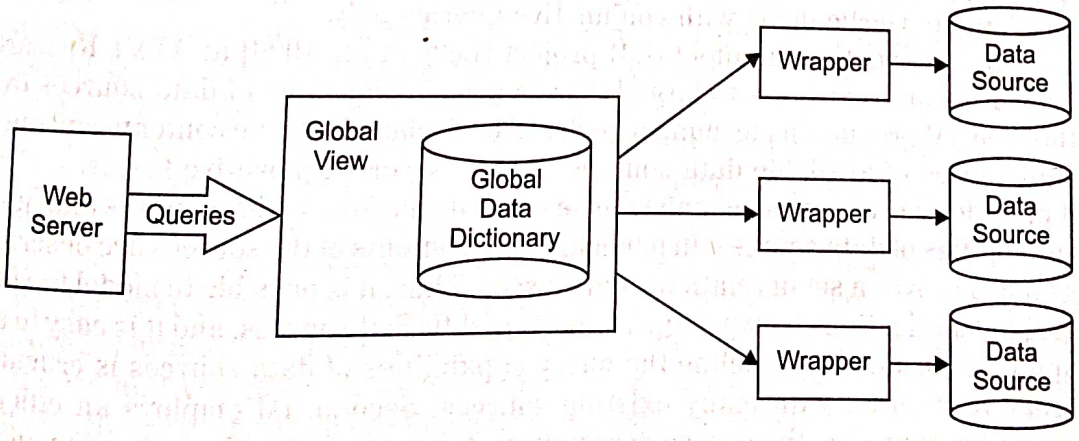

Figure 16.10. Hierarchy of Mediators

16.3.5 Research Projects and Open Issues

There have been many research projects on data integration in the context of the Web. We only briefly mention a sample of them that have been well prototyped and demonstrated: Tsimmis at Stanford University (USA), Garlic at IBM Almaden Research Laboratories (USA), Information Manifold and Strudel at AT&T Research Laboratories (USA), and Disco at Inria (France). We conclude with some open issues.

Tsimmis [Hammer et al., 1995b] (The Stanford-IBM Manager of Multiple Information Sources) is a DARPA-funded joint project of the Stanford database group and the IBM Almaden database group for integrating semistructured Web data. Tsimmis follows the mediator-wrapper architecture, allowing for hierarchies of wrappers and mediators. The components communicate using the OEM semistructured data model and an associated query language called MSL (Mediator Specification Language). MSL statements are Datalog-like logical rules that can deal with objects. Tsimmis stresses the automatic generation of wrappers and mediators using a context-free grammar that facilitates the description of query capabilities. Tsimmis focuses on the optimization of select-project queries, with little attention to joins, which are considered unlikely in the Web context.

Garlic [Haas et al., 1997], [Tork Roth and Schwarz, 1997] is an information integration project of the IBM Almaden database group. Garlic assumes that the query capabilities of the sources are unknown to the mediator, and finding an execution plan amounts to negotiating with the sources as to how much of a plan they can handle. The Garlic strategy may lead to unnecessary network traffic between the mediator and the wrappers, but it is powerful enough to deal with virtually every kind of conjunctive query.

Garlic provides an object-oriented view of the data sources, modeling all collections of similar items in class interfaces. For each interface, there may be more than one implementation, each of which corresponds to some subset of that collection, located in some data source. The wrappers, too, are objects: their methods correspond to supported operators. Query planning is done by invocating the wrapper methods and deducing from the return result how much the wrapper can handle. Garlic deals with conjunctive queries only.

The Information Manifold (IM) project [Levy et al., 1996] at AT&T Research provides uniform access to large heterogeneous collections of data sources over the Web. IM provides a mechanism to describe declaratively the contents and query capabilities of available data sources. IM has several innovative features. First, it provides a practical mechanism to describe declaratively the contents and query capabilities of data sources. In particular, the contents of the sources are described as queries over a set of relations and classes. Thus, it is possible to model the fine-grained distinctions between the contents of different sources, and it is easy to add and delete sources. Modeling the query capabilities of data sources is crucial in order to interact with many existing sources. Second, IM employs an efficient algorithm that uses the source descriptions to create query plans that can access several data sources to answer a query. The algorithm prunes the sources that are accessed to answer the query, and considers the capabilities of the different sources.

The query planning algorithm is a variation on algorithms for rewriting queries using views. In addition, IM is able to reason about sources that are known to be *complete* [Levy, 1996] and with probabilistic information about the sources [Florescu et al., 1997]. [Ullman, 1997] provides a detailed comparison of IM and Tsimmis.

Strudel [Fernandez et al., 1998] is another subsequent information integration project at AT&T Research focusing on semistructured data. The Strudel system applies concepts from database management systems to the process of building Web sites. Strudel's key idea involves separating the management of the site's data, the creation and management of the site's structure, and the visual presentation of the site's pages. First, the site builder creates a uniform model of all data available at the site. Second, the builder uses this model to declaratively define the Web site's structure by applying a "site-definition query" to the underlying data. Third, the builder specifies the visual presentation of pages in Strudel's HTML-template language. The data model underlying Strudel is a semistructured model of labeled directed graphs.

Disco [Tomasic et al., 1997], [Tomasic et al., 1998] (Distributed Information Search Components) is a data integration project at Inria, Rocquencourt. Disco's architecture is based on the three-tier architecture, extended with several novel features.

Disco mediators and wrappers operate independently: a mediator accesses a wrapper simply through a URL-like description of the wrapper. This feature means that wrappers can easily be shared among multiple mediators. Each Disco wrapper exports its capabilities using a grammar-like description of the operations that the wrapper supports. Disco mediators automatically adapt to the capabilities of wrappers by using an elegant distinction between the preliminary execution plan (which does not consider wrapper capabilities at all) and the final execution plan, which accounts for wrapper capabilities [Kapitskaia et al., 1997]. In addition, wrappers optionally export *cost statistics* and *cost equations* that describe the size of the data in the underlying sources and the cost of accessing the sources [Naacke et al., 1998]. Disco mediators use this cost information to perform sophisticated cost-based query optimization.

Disco mediator query processing can continue to function when some data sources are unavailable. During query processing, unavailable data sources are detected. Query processing continues with the available data sources by saving the partial results of these sources. When the unavailable data sources become available, their results are integrated with the saved results to produce the answer to the saved partial results can be examined by applications through the use of secondary queries, called *parachute queries* [Bonnet, 1998].

As the projects described above suggests, research on information integration on the Internet is very challenging and is only beginning. These projects have improved sources. But much more work is needed to ease the deployment of mediators and wrappers in various application domains. Besides dealing with semistructured data, there are important issues, such as mixing IR and database capabilities, dealing with data source failures, updating either integrated or source data, and improving the overall data quality.

16.4 PUSH-BASED TECHNOLOGIES

The push-based approach to data delivery and dissemination is a response to some of the problems inherent in pull-based systems. One of these problems is that users need to know a priori where and when to look for data. When data volatility is high, users must hunt for new data frequently, wasting time and effort [Franklin and Zdonik, 1998]. A second problem is the mismatch between the asymmetric nature of some applications (we listed some of these in the introduction) and the symmetric communications infrastructure on applications such as the Internet. There are, of course, asymmetric communication media, such as the cable TV network and satellite-based systems, but an increasing amount of data communication is being done over the Internet, for which this problem exists.

A number of different types of asymmetry can be identified [Franklin and Zdonik, 1997], each having an impact on the data delivery mechanism. The first, and most obvious, is network asymmetry, where the network bandwidth is different in the upstream (from clients to server) than the downstream (from server to clients) channels. This type of asymmetry is exhibited in mobile networks, satellite-based systems, cable TV networks, and even telephone networks with the emerging ADSL technologies.

A second type of asymmetry arises in distributed information systems, due to the imbalance between the number of clients and the number of servers. Typically, the number of clients is larger than the number of servers, but if this ratio becomes too high, it causes delays and difficulties in data delivery. Server overload is a well-known problem (thus the nickname of the WWW as the World Wide Wait).

The third type of asymmetry is the result of differences in the amount of data transmitted between servers and clients. For example, traditional Web search engines transmit only a few keywords upstream, but the downstream transmission of the research results may involve significant amounts of data.

Data volatility is the source of a fourth type of asymmetry. If the data and services that clients wish to access change frequently (data may be updated, or new services may be provided), there will be an asymmetry in the "control" of data flow, since the servers will mostly control the action.

In environments which exhibit one or more of these asymmetries, pull-based data delivery may not yield good results. The alternative is to resort to push-based techniques, which resolve the problems faced by pull-based systems in meeting the requirements of these environments.

A fundamental difficulty with the push-based approach is that the control of data delivery is transferred from the clients to the servers. Clients no longer request data when they need it; data are delivered to them when the server transmits them. This raises a number of issues. The first is the generation of a data transmission schedule that meets clients' needs. A second, related problem is managing client caches so that the average wait time to access data is minimized. Finally, there is the problem of propagating data updates to all the clients. We investigate these problems within the context of a particular push-based approach called "broadcast disks" [Acharya et al., 1995], [Acharya, 1998].

16.4.1 Delivery Schedule Generation

Data delivery schedule generation arises as a problem in push-based systems because clients no longer pull the data they need whenever they need them. Thus, the servers have to determine when to push which data to the clients. A straightforward method is to broadcast the data when it becomes available. At steady-state, this amounts to transmitting data continuously and repeatedly at fixed intervals. This approach is similar to a disk which spins at a fixed rate, from which clients read the relevant data as they pass by[3]—hence the name "broadcast disk." When each data item is broadcast at the same, fixed interval, it is referred to as a flat disk. Figure 16.11a la depicts a flat disk schedule, where data items A, B and C are broadcast in the order A-B-C.

If the applications access data items at fixed intervals, then the flat disk is the optimal approach. If application access to data is not uniform, then flat disks do not perform well. It is preferable to transmit pages that are more in demand (*hot pages*) more frequently than those which are demanded less frequently (*cold pages*). There are two alternative schedules in this case: skewed and multi-disk. In skewed broadcast (Figure 16.11b), the subsequent broadcasts of each hot page are spaced randomly in the schedule. In the example of Figure 16.11b, this results in the schedule A-A-B-C, since A is the hot page and is transmitted twice as frequently as B and C. In the multi-disk approach (Figure 16.11c), hot pages are broadcast more frequently, but at regular intervals; that is, the inter-arrival time of each page is fixed. This is similar to placing data on multiple disks with varying speeds—hence the name. Studies indicate that the multi-disk approach gives the best performance in cases where application access to data is non-uniform [Acharya et al., 1995].

The question now is how to generate a multi-disk schedule. The input to a schedule generation algorithm is the applications' (clients') access pattern to data. Determining this is not easy. One possibility is for the clients to post their access profiles—similar to what happens in a publish/subscribe-based system. Given this information, the schedule generation problem can be formulated as a mathematical optimization problem that allocates bandwidth to different data items, such that the expected delay for a data item is minimized. We leave the formulation as an exercise for the readers, and instead describe an heuristic algorithm proposed in [Acharya et al., 1995] and [Acharya, 1998].

The algorithm generates a periodic schedule, in which the inter-arrival times of subsequent transmissions of a data item are fixed. With these constraints, the algorithm allocates as much of the bandwidth as possible in order not to waste any resources. The steps of the algorithm are as follows:

1. Order the data items from hottest to coldest.
2. Partition the data items into ranges of items, such that the items in each range have similar application access profiles[4]. The number of ranges is denoted by *num_ranges*.

[3]Note that all the clients "hear" the same data page at the same time.
[4]Each range corresponds to a disk of a particular speed.

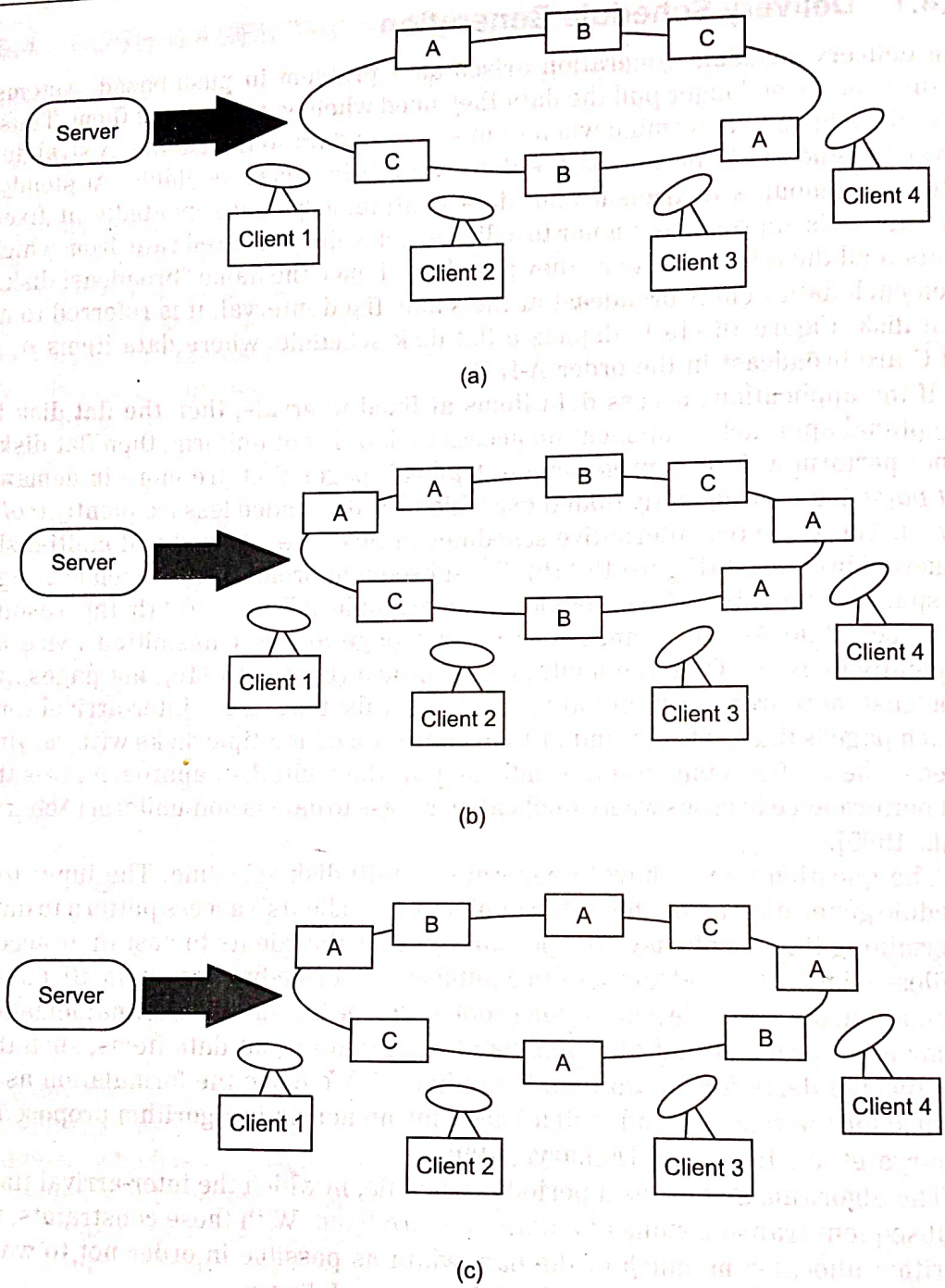

Figure 16.11. Alternative Broadcast Disk Schedules

3. Choose the relative broadcast frequency for each range as integers (rel_freq_i, where i is the range).
4. Divide each range into smaller elements, called *chunks* (C_{ij} is the j-th chunk of range i). Determine the number of chunks into which range i is divided as $num\_chunki = max\_chunks/rel\_freq_i$, where *max-chunks* is the least common multiple of rel_freq_i, $\forall i$.

5. Create the broadcast schedule by interleaving the chunks of each range using the following procedure:

```
for i from 0 to max_chunks − 1 by 1 do
    for j from 1 to max_ranges by 1 do
        Broadcast chunk C_j, (i mod num_chunks_j)
    end-for
end-for
```

Example 16.1

We demonstrate the algorithm using an example taken from [Acharya et al., 1995], [Acharya, 1998]. Consider 11 data items $D_1, \ldots, D_{11}$, where D_1 is the hottest and D_{11} is the coldest. Assume that data item D_1 has one access pattern, items D_2 and D_3 have another access pattern, and items $D_4, \ldots, D_{11}$ have a third. Thus, we have three ranges (Figure 16.12). Let us assume that, based on the access profiles, it is decided to transmit range 1 twice as frequently as range 2, which should be transmitted twice as frequently as range 3. Thus, $rel\_freq_1 = 4$, $rel\_freq_2 = 2$, $rel\_freq_3 = 1$.

In the fourth step of the algorithm, these ranges are subdivided into chunks. The least common multiple of *rel_freq's* is 4, from which we obtain $num\_chunk_1 = 4/4 = 1$, $num\_chunk_2 = 4/2 = 2$, $num\_chunk_3 = 4/1 = 4$. Step 5 of the algorithm creates a periodic broadcast schedule in which each

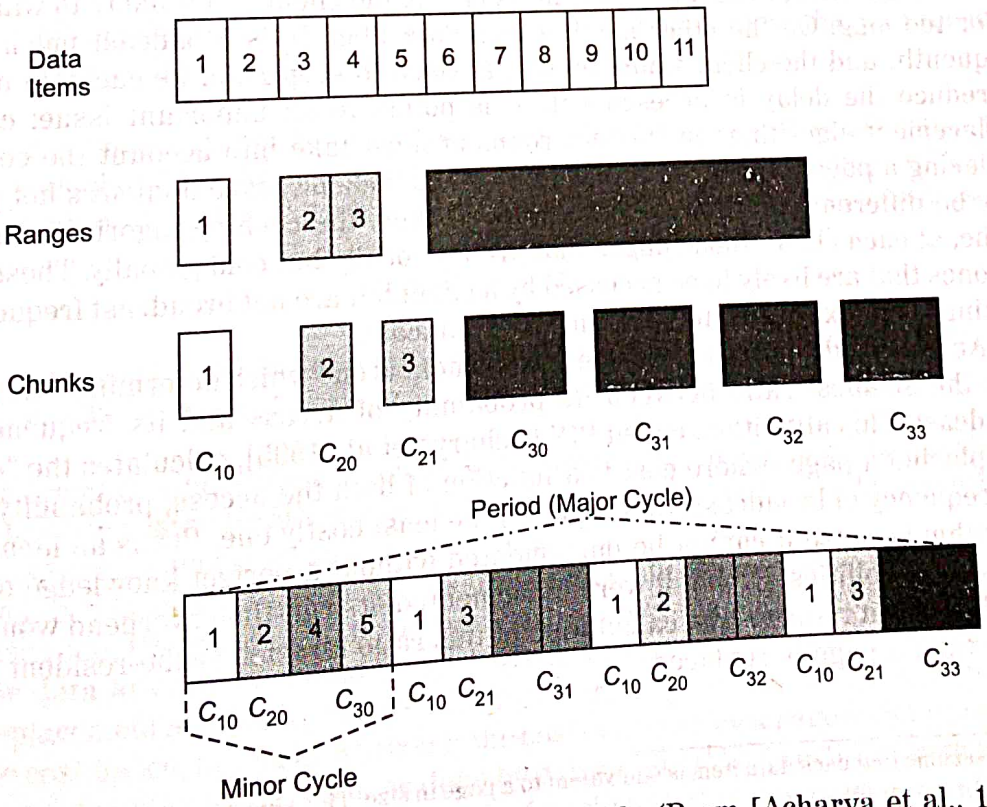

Figure 16.12. Example Broadcast Schedule (Prom [Acharya et al., 1995])

period (or major cycle) consists of four sub-periods (or minor cycles), where each minor cycle contains one data item from a different chunk. Thus, the inter-arrival time for each data item is fixed. The resulting schedule is depicted in Figure 16.12.

16.4.2 Client Cache Management

The broadcast schedule is determined by the server based on the access profiles of many clients. Thus, the schedule represents a compromise between the access patterns of many clients. Furthermore, the server may give higher priority to the requests of some clients over others. Since the bandwidth is fixed, improving the delay for one client inevitably causes a degradation in the delay for another client. Client profiles may also change over time, causing schedules that were once optimal to cease being optimal after a while. Thus, it is important for clients to manage their caches so as to keep as many of the needed data items as possible in their caches.

There is a difference between pull-based and push-based systems in the cache management algorithms that are employed. These differences are exhibited in both the demand-driven cache replacement policies and the prefetching mechanisms, We discuss both in the remainder.

If traditional caching policies (such as LRU) are used, then each client caches its hottest pages. However, this may not be optimal. Consider a case where D_i is the hottest data item. This means that D_i will be broadcast more frequently than other items. Thus, even if D_i is not in its cache, the client is not likely to wait for it for too long. On the other hand, a cold data item D_j is broadcast much less frequently, and the client would be well-advised to keep D_j in its cache in order to reduce the delay in accessing it. This points to an important issue: cache replacement algorithms in this environment must take into account the cost of replacing a page[5]. This cost is different for each client, since a client's hot page may be different from the global hot page. Thus, the caching algorithm should cache, at each client, those pages that are hot locally but cold globally. These are the ones that are likely to be accessed by a client but are not broadcast frequently, making them expensive to get from the broadcast.

An idealized algorithm for page replacement is one which determines the page with the smallest ratio between its probability of access and its frequency of broadcast. This algorithm, called PIX [Acharya et al., 1995], calculates the "cost" of replacing a page (where cost is a function of both the access, probability and the frequency of broadcast) and replaces the least costly one. PIX is an idealized algorithm because it cannot be implemented without a perfect knowledge of the access probabilities. Even if it were implemented, its run-time overhead would be very high, as it requires the calculation of this ratio for each cache-resident page every tune a page is replaced.

[5]We assume that each data item is equivalent to a page in size. This simplifies the description of the issues.

An implementable approximation of PIX, called LIX, is described in [Acharya et al., 1995]. LIX maintains a number of linked lists of cached pages, one per range (or disk) that is involved in the broadcast schedule. When a page is brought into the cache, it is always inserted into the chain that corresponds to the range (disk) to which the page belongs. LIX can flush out a page from any of the linked lists to open up space for the incoming page. Thus, the sizes of the linked lists dynamically change as pages come in and out of the cache. As in the well-known LRU algorithm, LIX orders the pages in each chain, such that the pages that are most recently accessed are at the top of the chain, and those that have been least recently used are at the bottom. To control this action, the algorithm maintains two variables for each cached page: (a) Pr_i, which is the probability of access of page P_i, and (b) LT_i, the time of the most recent access to page P_i. The operation of the algorithm is as follows:

1. When a page P_i is brought into cache and inserted into a chain,

$$Pr_i = 0$$
$$LT_i = CurrentTime$$

2. When P_i is accessed again, it is moved to the top of its own chain and the following calculations are made:

$$Pr_i = HF/(CurrentTime - LT_i) + (1 - HF) * LT_i$$
$$LT_i = CurrentTime$$

HF is called the history factor and is a constant that weighs the most recent access with respect to the running probability estimate.

3. If a new page needs to be flushed out to open up space, a *lix* value is calculated for the pages at the bottom of each chain and the page with the lowest *lix* value is flushed out. The *lix* value is calculated as follows:

$$lix_i = Pr_i/rel\_freq_i$$

where rel_freq_i is the relative broadcast frequency of the range (disk) to which that page P_i belongs.

Performance studies demonstrate that a page replacement algorithm, such as LIX, is superior to the traditional algorithms, such as LRU, in terms of the average response time to access data items [Acharya et al., 1995].

A natural issue to consider next is prefetching and its impact on the average response time. Prefetching in the broadcast disk environment is cheap; the client gets the data anyway, so would be well-advised to grab it. As in the case of the page replacement algorithm, the prefetching algorithms in a broadcast disk system are also cost-based, taking into account the cost of accessing a page in determining which page to prefetch. This can be done by determining the "worth" of each page when it is broadcast, and if its worth is higher than that of any page currently in

the cache, a prefetch (and a consequent page flush-out) is performed. This value for a page P_i can be computed as $pt = Pr_i * tr$, where tr is the time remaining before P_i's next broadcast. This is the basis of the PT algorithm [Acharya et al., 1996a].

The pt value of a page is dynamic, since one of its components (tr) changes with each clock tick, pt is highest at time t_i, when it is broadcast. From there on, it steadily reduces until the next broadcast at time t_j, when it shoots up again. The result is a sawtooth-shaped pt curve. This points to a fundamental difference between the demand-based page replacement algorithm PIX and the prefetching algorithm PT. When PIX is used as a prefetcher, the cache will fill up with the pages with the highest PIX values, and since these are static, prefetching will stop when the cache is full. In contrast, since pt values of pages change with every tick, the contents of the cache will continually change.

PT is also an idealized algorithm and is impractical to implement, since it requires that the pt values of *all* cached pages be computed at every clock tick—clearly not an acceptable overhead. The solution is to follow the example of LIX and divide the cached pages into ranges, such that each range has the same access probability. This ensures that there is only one candidate victim for replacement for each range, thus reducing the number of pt values that must be calculated. This approximation of PT, which is called APT, maintains a doubly-linked circular list for the pages in each range. Furthermore, for each range, it maintains a pointer (*PRV*) to the page with the smallest tr; i.e., the page that is that range's potential replacement victim. When a page must be flushed out, APT calculates the pt values for only the *PRV* pages in each of the ranges and chooses the one with the smallest value. The freed-up page is allocated to the range into which, the new prefetched page is to enter.

As expected, prefetching can provide significant performance improvements over demand-based caching. In this case, experiments reveal a 10–30% improvement in favor of APT over LIX [Acharya et al., 1996a]. In push-based environments, a fundamental problem with prefetching—namely, the additional load it places on system resources—does not exist, since prefetching is controlled locally by each client according to his/her own access profile.

16.4.3 Propagating Updates

Updating a database means adding, deleting, or modifying data. In a push-based system, these roughly correspond to adding and deleting pages from the broadcast schedule, or modifying page contents. Adding a new page to the schedule is, perhaps, the least problematic, since the server can compute the new broadcast schedule and inform the clients of the changes. Deletions and modifications are more problematic.

Some of the difficulties arise from the fact that a push-based system, as we have described it in this section, is a data shipping system in which the server broadcasts data and the clients cache them. Thus, the issues that we discussed in Chapter 14 regarding client cache management apply here as well. One of those problems, to remind ourselves, is notifying clients of deletions and modifications. In the case of deletions, an invalidation approach is applicable where the server

informs the clients that the pages they have in their caches are no longer valid and should be flushed. In the case of data modification, the server may either invalidate the modified pages in clients' buffers or propagate the changes and ask the clients to apply these changes to the pages in their buffers.

In the case of data modification, there are additional problems related to the consistency of the data. The issue was discussed in Chapters 10–12 within the context of regular database transaction processing. In push-based systems, a number of differences arise. One difference is, again, the result of the data shipping nature of the system: how does an update that originates in one client get propagated to other clients? This particular issue was also discussed in Chapter 14 within the context of client cache management. A second problem is the consistency constraint that is used and its possible effect on the propagation of updates. A number of possible consistency models can be identified [Acharya et al., 1996b]:

- *Latest value*: Clients are required to access the most recent value of a data item.
- *Quasi-caching*: Clients can access data items which deviate from the latest value according to a tolerance that is defined individually for each client [Alonso et al., 1990]. This tolerance may be temporal or value-based, or it may be defined otherwise.
- *Periodic*: Data values change at periodic intervals (e.g., at the beginning of minor or major cycles in the broadcast schedule).
- *Serializability*: Clients' data access schedules are required to result in a seri-alizable schedule.
- *Opportunistic*: Clients access whichever version of the data they can find.

It is clear that some of the consistency models will affect the broadcast schedule and the way in which caches are utilized. For example, if *latest value* is the accepted consistency definition, then either clients should not cache any data and wait for the server to broadcast updates immediately, or an algorithm must be devised which would allow the clients to cache data, but allow them to "back-up" if the values they have read are not the latest values (perhaps determined by timestamping both the server notifications and the client-cached data).

The problem of propagating updates in a push-based system is not well understood. There is only one study at this time [Acharya et al., 1996b] that has analyzed the problem in a relatively restricted setting, where all the updates are collected at the server and clients do not have any opportunity to communicate with the server (they just listen to the broadcast). Clearly, more studies are required in this area.

16.5 MOBILE DATABASES

In Chapter 3 we discussed wireless networks and their characteristics (Section 3.5, to be precise). In this section we discuss some of the issues that arise in building

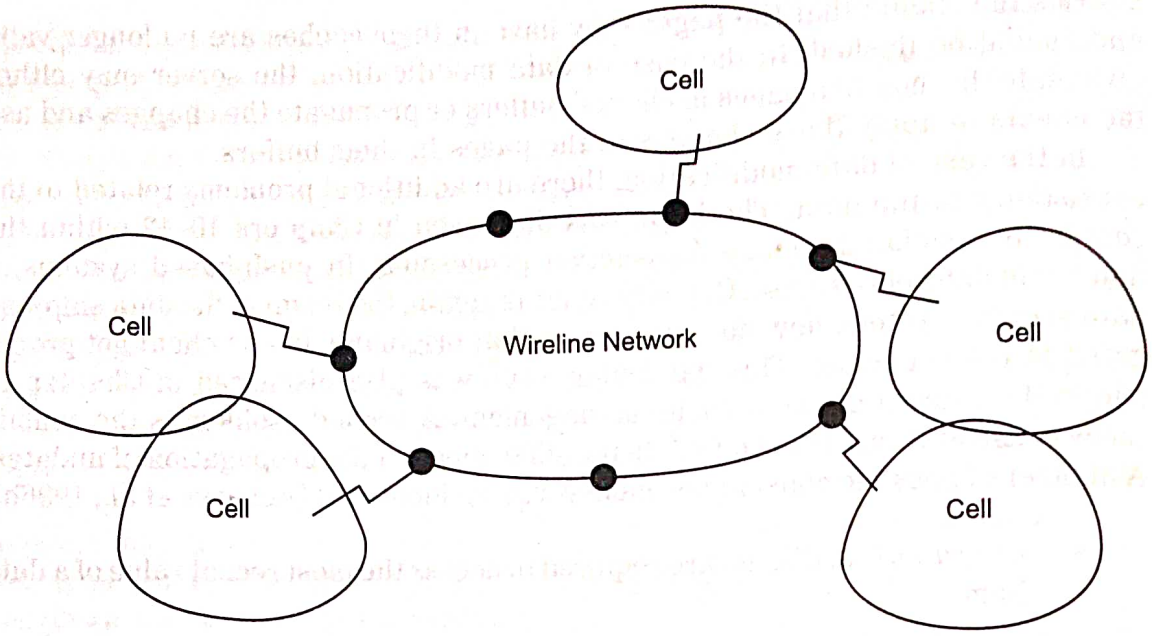

Figure 16.13. Cellular Networks

DBMSs over such networks. Since the discussion in Section 3.5 is far removed from this one, we start with a summary of our discussion in that section.

A wireless network consists of a "wireline" (fixed) backbone network on which a number of control stations are located. Each control station coordinates the communication from a mobile computer [also called *mobile station* (MS) or *mobile unit* (MU)] in its respective cell to another mobile computer in the same cell, or in another cell, or to a stationary computer on the wireline network. This is depicted in Figure 16.13 (which is the same as Figure 3.13).

In such an environment, data can reside on either the fixed network or the mobile stations. In this section, we are fairly flexible in our definition of a mobile database—we use the term to refer to any arrangement in which the database access is performed by mobile stations over a wireless link.

There can be a number of different types of mobile stations. One type involves fairly simple computers with limited capabilities. In this case, data are located in computers on the wireline network, with the mobile stations "downloading" data as they need them. This scenario is realistic for some applications, and is currently the most common. However, the distributed data management problem is not significantly affected by mobility because data resides primarily on wireline computers. More interesting is the environment in which the mobile stations are more powerful and store native data that may need to be shared by others—the so-called "walkstation" case [Imielinski and Badrinath, 1994]. It is this latter class of devices, which can hold data and process them, that raises the interesting questions.

Mobile computing environments are characterized by three issues [Forman and Zahorjan, 1994]: communication characteristics, mobility and portability. Communication is conducted over wireless networks which are prone to disconnections,

noise, echo, and low bandwidth. Mobility of some of the equipment on the network causes static data in stationary networks to become dynamic and volatile in wireless networks. Mobility raises issues such as address migration, maintenance of directories and difficulty in locating stations. Finally, portability places restrictions on the type of equipment that can be used in these environments. For example, easy portability and the desire for long operation between battery recharges usually restrict the type and size of storage that can be used.

These characteristics of mobile environments can be expanded [Alonso and Korth, 1993], [Dunham and Helal, 1995], [Imielinski and Badrinath, 1994], [Pitoura and Samaras, 1998] as follows:

1. The wireless networks have restricted bandwidth. They can be either wireless cellular networks, which have a bandwidth in the order of 10 Kbps, or they can be wireless local area networks with a bandwidth of 10 Mbps. The former can cover larger geographic areas, while the latter is restricted to smaller areas, typically within a building.

2. The power supplies (i.e., batteries) in mobile stations have limited lifetimes. Even with the new advances in battery technology, the typical lifetime of a battery is only a few hours, which is reduced further with increased computation and disk operations. This problem is not likely to disappear in the near future, since battery power is expected to increase by only 30% over the next five years.

3. Because of power restrictions, mobile stations are not available as widely as stationary ones. They are usually powered off when they are not in use. For the same reason, the amount of computation that can be performed on mobile stations is restricted.

4. As the name suggests, mobile stations move in these systems. This requires additional system functionality to track them, in particular if they store data shared by others. The mobility also necessitates the management of heterogeneity of the base stations, since they can have quite varying capabilities. Finally, station mobility causes constant change in the topology of the underlying network.

These issues raise interesting questions for the design of distributed DBMSs, since they cause significant changes to the infrastructure upon which such systems are built: almost all the DBMS functions are affected to varying degrees. In the remainder, we highlight these issues.

16.5.1 Directory Management

Directory management is very much linked to distribution design. The problem here is the "optimum" distribution of data across the nodes of the wireline (stationary) network, as well as data that can reside on the walkstations. As discussed in Chapter 5, this is a problem in stationary networks, and it becomes more difficult

in mobile environments due to the mobility of the walkstations (which move resident data) and other mobile stations. The typical optimality measures take into account who is trying to access data from where, and distribution algorithms attempt to place some or all of that data in close proximity to the access points. These measures and the related design arguments must be revisited in mobile environments.

The fundamental question is how to locate a mobile station which may hold the required data. Two solutions have been discussed in the literature [Imielinski and Badrinath, 1994]. In one solution, each mobile station has a "home" base station which keeps track of its location by receiving notifications of its movements. Thus, any other station (say, station A) which wishes to find this mobile station (say, station B) contacts B's home base, where it learns the current location. This method, which is currently used in cellular networks, is simple, but may cause significant access latency if the mobile station is far from its home base. The second solution is based on restricted broadcast within the area of the (mobile or stationary) station which wishes to access the mobile station in question. In this scheme, station A, which wishes to establish access, first broadcasts an inquiry message to the base stations in its own neighborhood to see if B is in that area. If this does not yield a positive result, then A contacts B's home base. A number of other alternatives are possible, of course, even though they are not mentioned in the literature. For example, if a mobile station knows its itinerary, the directory information can be updated in advance to improve the chances of success in the second algorithm. Another alternative may be for mobile stations to leave behind their new addresses as they move, allowing them to be tracked (similar to the object migration techniques discussed in Section 14.4.3). This, coupled with the caching of location information at the accessing stations, may reduce the directory search latency at the expense of movement tracking.

A second directory management problem is finding the optimum location of data when the accessing unit is moving. If data exists at multiple sites, the locations where data access occurs when the accessing station is moving becomes important. This is related to query optimization, which we discuss in Section 16.5.4; however, an important component of the problem is directory management, since the data must be located first.

16.5.2 Caching

Many of the applications of mobile information systems (e.g., traffic monitoring and advice, sales and customer tracking) involve caching data at the mobile station for a period of time. As indicated above, the limited power supply at the mobile stations also restricts the amount of processing that is performed at the mobile stations, requiring the computation to be done on stationary stations and the resulting data to be sent to the mobile stations. This is a typical example of a query shipping system. More powerful mobile stations may also participate in data shipping, where an entire DBMS resides at the mobile station. As such, issues related to the management of the caches at mobile stations arise. A thorough

discussion of caching issues such as fetching, coherency and replacement can be found in the literature [Pitoura and Samaras, 1998].

In the mobile environment, caching granularity becomes a crucial issue. Trade-offs between page and object caching were discussed in Chapter 14. Both of these issues can create severe performance problems due to the limited bandwidth of wireless computing. Semantic caching has been proposed as a solution to overcome some of these performance problems associated with traditional granularities [Dar et al., 1996b] [Keller and Basu, 1996]. With semantic caching, the granularity of cached data is the result of a query. The contents are defined based upon a predicate indicating the selection criteria. The benefits of semantic caching to mobile computing are the reduced network traffic and reduced space requirements for caching at the mobile station. In addition, semantic caching easily supports replacement strategies based on location. As the mobile unit moves, data associated with certain remote locations (see the discussion of location dependent data below) may become obsolete and need to be replaced. Semantic caching supports this type of cache management strategy. In addition, semantic caching may prove to be beneficial in supporting disconnected operation. This is due to both the reduced space requirements for the cache and an increased hit ratio because of the semantic similarity of the client's queries.

Cache coherency of the mobile station cache has been studied extensively. Updates at the server may either be propagated to the mobile stations, or mobile station caches may be invalidated. Simulation studies have shown the effectiveness of both types at reducing the uplink traffic on the wireless link [Cai et al., 1997].

16.5.3 Broadcast Data

Broadcasting data from base stations to mobile ones can improve the performance of mobile applications. All the issues we discussed in the previous section on push-based technologies apply in this case as well. One particular constraint is that, since the mobile stations power off frequently to conserve energy, it is necessary to either have a very regular broadcast schedule (so that they can "wake up" at the right time to receive relevant data) or have the data be self-indexed [Barbara and Imielinski, 1994], [Imielinski et. al., 1994].

Due to the mobility of the mobile stations, the content of any data to be broadcast should be dynamic and adaptive. One approach proposes cooperation between the clients and servers [Datta et al., 1997]. A performance metric called *tuning time* was used to evaluate how long a client must listen to a channel. Periodic broadcast performed best at high loads, while an aperiodic approach was best at a lower workload. The air-cache approach dynamically changes the content of the broadcast disk based on the access frequency of the data [Stathatos et al., 1997]. Changes in access frequency change the content of the data that is broadcast. The mobility factor again complicates this issue. As the mobile station moves from cell to cell, the client's need for data moves with him/her. Thus, the content of the data broadcast should adapt to this movement. As the user moves from one cell, the data needed only by that user should be dropped from the content of the

data broadcast in the old cell. Also, it should be added to the data broadcast in the new cell.

16.5.4 Query Processing and Optimization

Query processing is one of the DBMS functions affected most by the mobility of the environment. The effects are both in terms of the queries that are posed and the optimization techniques that can be used.

Queries in this environment can be "location dependent." For example, the query "What are the names and addresses of French restaurants in this city?" returns different results depending on which city the mobile station is located in. Location dependence may be more sophisticated, involving the tracking of moving objects. For example, if a user asks the query "Where is a branch of XYZ bank within 1 kilometer of where I am?" while driving a car, the system has to track the movement of the car in order to answer the query. There are cases in which it is even necessary to track the movement of multiple objects. There is some work in this area [Sistla et al., 1997], [Li et. al., 1997], but it is still a largely unresolved problem.

In traditional DBMSs, only the characteristics of the processing node are taken into consideration for managing data, not its physical location. The traditional approach thus promotes "location transparency". The need to support location dependent queries, however, changes this requirement in the mobile computing arena. Location Dependent Data (LDD) are data whose values are determined by its location [Dunham and Kumar, 1998]. For example, data about local TV stations' affiliation with national networks will vary from city to city. LDD can be used to answer location dependent queries. With this approach, the same query stated in different locales will obtain different results because the data values themselves are different. These multiple "correct" values of data produce a new type of data replication based on location called spatial replication. Different spatial replicas of the same data object may have different values because they are associated with different locations. LDD complicates caching functions. Cached data can become stale not because of the update of the data at the server, but because the mobile unit has moved into a new region where the cached data is not valid.

Location dependent queries can be processed by augmenting each query with location information, by assuming that queries are not modified but that location dependent data are used, or by a combination of these two approaches. Regardless of the approach used, when a query is requested, it must be bound to a location and a set of data values. Since the mobile unit is moving, the query could be bound to different locations: the location of the mobile unit when the query was requested, the location of the mobile unit when the query terminates, a projected location of the mobile unit based on its current movement, or a location as specifically indicated in the query.

With respect to query optimization, the mobility of access stations makes it difficult to determine communication costs between the accessing station and the station where the data resides. Furthermore, if the requested data is stored on one of the walkstations, then both network nodes may be in motion. The low bandwidth

and, more importantly, the variability of the bandwidth between the wireless part and the backbone, complicate the picture further.

In a centralized database environment, query optimization usually involves determining the best approach, among a set of alternatives, to process a query which minimizes I/O cost. In a distributed environment, the dominant cost is usually viewed to be that of network traffic. While I/O and network costs are still important, in a mobile computing environment the mobility and dynamic state of the mobile unit complicate optimization even further. The contents of the cache may change dynamically. As the mobile unit moves, data may or may not become available via a broadcast disk. The best location from which to access data on a database server in the fixed network may change as the mobile unit moves. Thus, in the mobile computing environment, the plans and costs associated with various plans change based on the movement. Static optimization strategies are not appropriate. Dynamic strategies which adapt to the changing environment are required.

16.5.5 Transaction Management

The disconnection of stations for possibly long periods of time and bandwith limitations require a serious reevaluation of transaction model and transaction processing techniques. Typical concurrency control techniques rely on locking. Since wireless networks are more failure-prone than their stationary counterparts, locking may not be a good solution. Locks on data items at a failed station may be held for a long tune, blocking the termination of a transaction. If that transaction holds locks at other sites, this would reduce the availability of data. It is necessary to consider different consistency criteria, as well as algorithms and techniques to enforce them. For the same reason, atomic commitment protocols such as 2PC may not be suitable, since the disconnection of one station may seriously reduce the availability of the database system. Therefore, it has been suggested that longer transaction models, coupled with data replication and lazy replication schemes, may be more suitable for the wireless environment.

There have been many proposals to model mobile transactions (e.g., [Chrysanthis, 1993], [Pitoura and Bhargava, 1995], [Walborn and Chrysanthis, 1995], [Yeo and Zaslavsky, 1994], [Dunham and Helal, 1997]) with different notions of a mobile transaction. The most common view seems to be a database transaction which is requested from a mobile station. Most of these approaches view a mobile transaction as consisting of subtransactions which have some flexibility in consistency and commit processing. Certainly a relaxation of the ACID properties is deemed necessary due to the anticipated high rate of disconnection. This disconnection and the associated limited battery power of the mobile units reflect themselves in a temporary suspension or failure of the mobile transactions. The management of these transactions may be static at the mobile unit or the database server, or may move from base station to base station as the mobile unit moves.

REVIEW QUESTIONS

16.1 Give a brief account of data delivery alternatives.
16.2 Give a brief account of data warehousing.
16.3 What do you mean by World Wide Web?
16.4 What do you mean by push-based technologies?
16.5 Give a brief account of mobile technologies.