DataX: an Approach to Ubiquitous Database Access

Hui Lei, Marion Blount and Carl Tait
IBM Thomas J. Watson Research Center
Route 134
Yorktown Heights, NY 10598
{hlei, mlblount, cdtait}@us.ibm.com

Abstract

This paper describes an approach for enabling remote database access from heterogeneous thin clients. DataX is a proxy-based architecture that supports disconnected operation by replicating a subset of the database on the mobile client, using a weak consistency criterion. It adapts data replication to device characteristics, link attributes and user preferences. It employs a per-device renderer to present data in a form layout, making the application logic separate from user interface logic and independent of the device type. It offers small footprint of client-side software, rapid development of end-to-end solutions, and portability across multiple server platforms.

1. Introduction

Many industries employ mobile workers who perform a significant part of their work at a customer location or away from the office. Examples of such mobile workers are insurance agents, visiting nurses and traveling salesmen. Many of the work processes in corporations have been re-engineered to give workers access to up-to-date data and the ability to update this data. In order to be productive, mobile workers need the same type of access to enterprise data.

Until recently, notebook computers and modems have been the main computing and communication resources for mobile workers. Thin clients, like palmtop computers and smart phones, have only been used for personal information management (PIM) applications. Thin clients' form factor, lower cost, and ease of use make them popular with users. Application developers are trying to exploit this by broadening the range of applications available for these thin client devices. Compared to desktop or notebook computers, thin clients are extremely resource-poor. They are operated in disconnected mode most of the time, either because there is no network connectivity available, or because the user wants to avoid the use of the network to save money or battery life.

The DataX project at the IBM T. J. Watson Research Center is developing middleware that enables database access from portable mobile computers, with special emphasis on thin clients. At this stage, we are focusing on data access scenarios where transactional properties are not required. Database applications such as data viewers and data collection fall into this category. DataX employs a three-tiered client/proxy/server architecture and supports disconnected operation through the following functions:

Subsetting: Selectively replicates portions of the server database on the mobile client.

Targeting: Adapts client-side replication based on device types, link attributes and user preferences.

Rendering: Presents the data in a device-specific manner for the user to view and manipulate.

Synchronization: Integrates the client's replica with the server's copy.

At a high level, our design goals are threefold. First, we want to effectively accommodate multiple types of portable computers, including thin clients. Second, we would like to make matters easy for both end users and application developers. Third, we want to be compatible with a corporation's investments in traditional technologies.

This paper is organized as follows. Section 2 discusses a number of considerations that influenced the design of DataX. Section 3 describes the DataX architecture in detail. Section 4 discusses related work in various areas. Our conclusions are presented in Section 5.

2. Design Rationale

Several interrelated factors have influenced our design decisions. These include the advent of thin clients, the need for interoperability and adaptability, and data consistency guarantees for applications. We examine these issues in this section.

Thin clients

The term *thin clients* refers to palmtop computers, personal digital assistants (PDAs), and smart phones that are emerging as a new wave of mobile computers. On one hand, their form factor, low cost, and ease of use are likely to cause the pervasive deployment of these devices for the appropriate set of applications. It is reasonable to expect that a significant number of new thin client users will be casual and unsophisticated. On the other hand, the same considerations of weight, power, size and economy exact a penalty in the hardware resources of these devices, including processor speed, memory and screen size.

As a result of the devices' limited physical resources and the users' naiveté, computational and administrative complexity should be reduced as much as possible on the client device. This can be achieved by the use of a proxy: an intermediary between the client and the server. A proxy usually has bountiful CPU cycles and is placed at the fringe of the fixed network [11]. Such an arrangement confers a number of advantages:

- The proxy can carry out intensive computation on behalf of the client. This in turn reduces the footprint of client-side code, making it possible for the client to host an application that would otherwise be too large. This also allows computation to be performed asynchronously on the proxy without the user waiting.

- The proxy is the ideal place for the system administration tasks associated with client devices. Mobile users can delegate these tasks to system administrators, who can implement them on the proxy. In particular, since some form of user assistance is essential to successful disconnected operation, all or part of that job can be handled by the administrator, alleviating the burden on end users.

- The servers can now concentrate on servicing the general user community, and not be concerned with the special demands and characteristics of mobile users. All data transformation and customization for mobile clients is handled at the proxy. Hence the services received by non-mobile clients are unlikely to be substantially compromised.

Interoperability and adaptability

The variety of database systems is impressive: there are currently as many as 30 different relational or hierarchical database systems. Client devices span the spectrum from laptops to smart cellular phones, with all kinds of PDAs, clamshells and palmtop computers in between. A company often has to support multiple types of servers and clients. The reasons are many: people bought what was cheapest, what was fastest, what was latest on the market, or what worked best for a single application. The diversity of data sources and user platforms calls for an open and interoperable system architecture.

Easy connectivity to multiple data sources can be achieved by relying on a standard database API such as Open DataBase Connectivity (ODBC) [6]. Supporting different types of client devices, each with a possibly different operating system, has lead us to two fundamental design decisions. First, data should be represented in a platform-independent format on the device. This will simplify the tasks of the proxy when sending data to and retrieving changes from the client. Second, the user interface logic needs to be separated from the data processing logic. It would be infeasible to require that application developers control the presentation of user interactions on a per device basis. This is particularly true when the data to be presented can be dynamically adapted, as explained below.

For a mobile user to have access to enterprise data while disconnected from the company network, he must hoard the necessary data on his mobile computer prior to disconnection, work with the local copy while disconnected, and integrate changes, if any, into the server upon reconnection. Three factors can affect the hoarding process. First, the characteristics of the client device may limit the amount of data that can be replicated on the device or make the presentation of certain data impractical. Second, the communication link between the client and the proxy determines the time needed to download the data. Third, different users may have their own preferences on how to make best use of their scarce hardware and time resources. Therefore, a flexible mechanism should allow for data replication and transmission tailored to client device type, link speed and user needs.

Data consistency

It is important to preserve data consistency between the partial replica on the client and the full database on the server. It is well recognized that a strong-consistency replication strategy is not suitable for mobile computing. Strong consistency ensures one-copy serializability, an intuitive and general-purpose consistency criterion. Unfortunately, enforcing strong consistency either incurs very high logging and repair costs, as in an optimistic strategy, or severely limits data availability, as in a pessimistic strategy [7]. For many applications, it is often acceptable to relax the data consistency guarantee. A weak-consistency replication strategy allows data accesses to be performed with little or no synchronization among replicas, only ensuring that replicas eventually reach identical states or become mutually consistent. Such a strategy is required to support disconnected operation of mobile computers because it allows for high availability, good scalability, and simplicity of the system.

In addition to mutual consistency, a database is often required to satisfy a set of integrity constraints on database objects and their attributes. These may range from semantic constraints such as arithmetic (in)equalities to referential integrity constraints. Best efforts should be made to enforce these constraints whenever the client replica is updated. This way, the user can be alerted to any anomalies at the first opportunity. All constraints are eventually enforced by the backend server when the client replica is reintegrated into the server's copy.

Another issue in weakly consistent systems is the user-perceived consistency [36, 13]. When data accesses involve multiple, potentially inconsistent, replicas, it is possible for a user to read or write to a data copy that is older than the copy he accessed before. Therefore it is desirable to present each user with a view of the database that is consistent with his own actions. Since a user in our system always accesses data through the local client replica, view consistency is trivially guaranteed within each disconnected session. However, view consistency across disconnected sessions depends on a user's actions. Since the backend database system is usually a standard system that provides strong data consistency, view consistency across disconnected sessions is guaranteed if the user always propagates all prior updates back to the server before hoarding data for a new disconnection.

A final issue concerns the degree of inconsistency between the secondary replica on the client and the primary replica on the server [1]. In a primarily disconnected mobile environment, this is again in the control of the user, who decides when to begin and end a disconnection. The local changes are propagated and the client copy refreshed at the discretion of the user. Such an operating mode is useful when the backend database does not change rapidly, or when stale data can be temporarily tolerated by the client.

3. System Architecture

Figure 1 shows the basic DataX system architecture. The system consists of three tiers: client, proxy and server. The proxy provides a single point of connection for heterogeneous types of client devices and performs the common subsetting, targeting, and synchronization functions on behalf on clients. The proxy bases its actions on a set of rules and data set definitions. The proxy accesses the server database through the ODBC interface so that the system is not bound to any specific server platform. Prior to a mobile user's voluntary disconnection, the proxy prepares necessary data as XML [39] documents and transmits them to the client device. During disconnection, the rendering component on the device controls the user interaction. The user may view the data on the device and perform updates. Upon reconnection, any changes made are propagated back to the proxy, again in the form of XML documents. The proxy is then responsible for integrating the changes into the server database.

We now describe the key components of the system in more detail.

3.1. Subsetting

The subsetting component extracts necessary data from the server database in preparation for a disconnection. Which subsets of data are needed is specified by the user. Although it reduces system transparency, such user assistance is essential for successful disconnected operation, since future user behavior is not perfectly predictable [23, 35]. For the mechanism to be practical, however, the user's participation should be limited at a high semantic level [26]. To that end, we institute an encapsulation mechanism called *folders*.

A folder is a parametric logical collection of data. A folder not only describes a subset of data, but also the operations allowed on the data. Folder definition is the responsibility of application developers. A mobile user specifies the data desired by selecting from a list of folder names, optionally providing parameter values to override defaults. The subsetting component on the proxy verifies that the user is allowed access to the requested folders and retrieves data accordingly.

A folder definition consists of the following four sections. Only the data section is directly used by the subsetting component; the other sections are needed by other components of the system.

1. *Header.* This section specifies the folder name, the parameters and their optional default values, and an indication of whether the primary key of the records can be *scanned* (looked up in a provided list). Information in this section will be used in the folder selection process. If the primary key is scannable, a list of primary key values, e.g., the names of an insurance agent's clients, will be maintained on the client device so that the user may request records by primary key.

2. *Data.* This section qualifies the records that belong to this folder. It is simply an SQL "**SELECT-FROM-WHERE**" clause and the **WHERE** condition is a predicate on the folder parameters. This data qualifier guides the data extraction on the proxy.

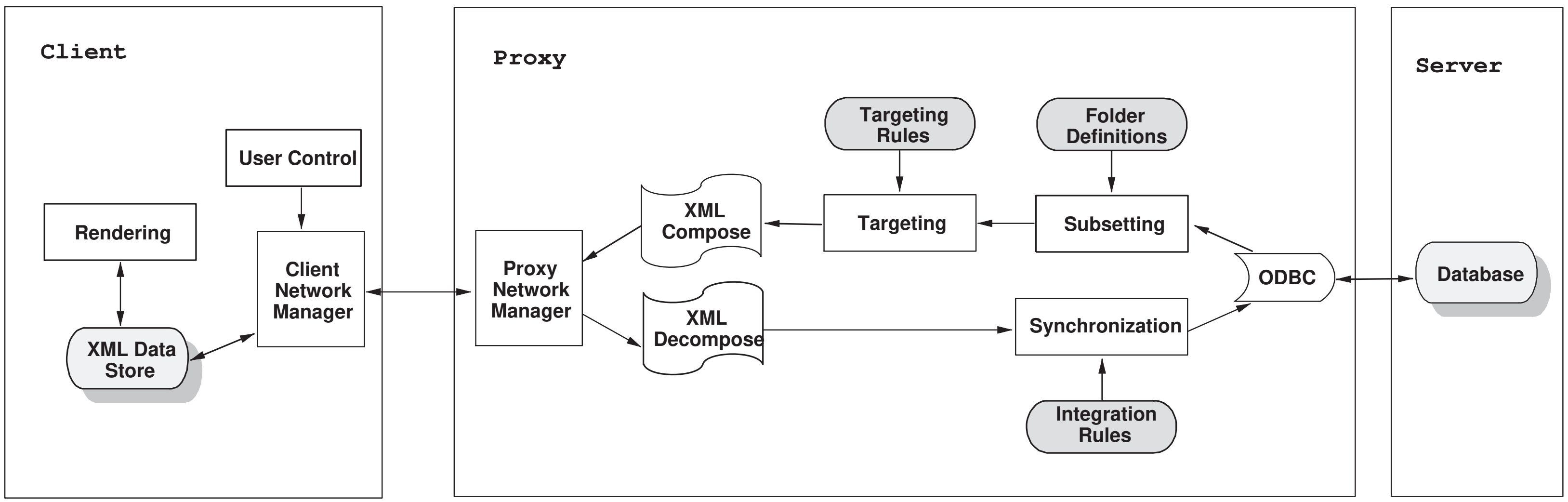

Figure 1. DataX Architecture

3. *Operations.* This section defines the operations a user is allowed to perform. At the record level, it specifies whether records are read-only or updatable and, if updatable, the operations permitted (insert, delete and/or modify). At the field level, it specifies whether each field is updatable and whether records can be sorted on the field. Information in this section is fed to the rendering component on the device so that the latter can effectively control the user interaction.

4. *Constraints.* This section specifies the integrity constraints for the folder. These are the constraints that are to be verified by the renderer whenever records are updated on the client. They are separate from the constraints that are part of the backend database definition and that are enforced by the database server.

3.2. Targeting

A folder determines the capability of its users: what data is accessible and how it can be accessed. However, data as stored in the server database may need to be transformed to make it appropriate for use in a mobile environment. Such transformation is the function of the targeting component. Reasons for data transformation may be that the destination device is not capable of hosting the complete data subset or rendering the data as it is, or that it takes too long to transmit original data over a slow link, or simply that the individual user prefers it so.

Roughly speaking, data transformation falls into one of two categories:

Filtering: Selective transmission of unmodified data.

Transcoding: Changing the fidelity and/or modality of a particular piece of data.

An example of filtering is transferring only some of the columns of a relational table. Truncating a text field beyond a certain size and changing a rich text into plain ASCII are classed as fidelity transcoding; converting text into speech is a form of modality transcoding. For semantic reasons, transcoding is only performed on read-only data.

Targeting is governed by a set of rules. Each targeting rule specifies an action and a condition under which the action may be triggered. The condition is a conjunction of one or more predicates, each of which tests one of the following:

Device characteristics. Currently we only support tests against device types (IBM WorkPad, ThinkPad etc.). Later we may allow tests on specific attributes such as screen size and color depth.

Communication attributes. The most useful attribute is the link speed.

Folder name. This is to allow folder-sensitive targeting. Both inclusion and exclusion tests can be specified.

Field name. A test on field name is valid only if it comes with an inclusion test on folder name.

Data type. A test on data type is usually an equality test to identify the applicable data type for a transcoding operation.

User ID. This can only be an equality test. This allows targeting to be customized on a per user basis.

Each targeting action is a data transformation function registered with DataX.

For common combinations of device and link attributes, DataX specifies default targeting policies. Application developers can introduce rules that are specific to an application (folder). Further, individual users may define rules that are tailored to their own needs. User-specific rules may or may not be application-specific. Here are some examples of targeting rules.

1. A general rule: *for a WorkPad, always convert rich text to ASCII.*

   DevType=WorkPad ∧ **Data**=RichText
   → Rich2Ascii

2. An application-specific rule: *if transferring customer records over a link slower than 56 kbps, truncate notes more than 100 characters long.*

   LinkSpeed<56 ∧
   Folder="CUSTOMERS" ∧
   Field="NOTES" → Truncate(100)

3. A user-specific rule: *when the CUSTOMERS folder is accessed by user SMITH on a WorkPad, filter out information on past customer visits (stored in the ACTIVITY field).*

   DevType=WorkPad ∧
   Folder="CUSTOMERS"∧
   User=SMITH → Exclude("ACTIVITY")

The targeting process for any data subset consists of two phases. In Phase 1, an applicable filtering rule, if found, is triggered. In Phase 2, for each data field, an applicable transcoding rule, if found, is triggered. In each phase, a rule is first searched for among user-specific rules, then among rules that are application-specific but not user-specific, and finally among general rules.

3.3. Synchronization

The role of the synchronization component is to modify the server's database to reflect any changes made on the mobile client and, optionally, to bring the client replica up to date. A change can be an insertion, an update, or a deletion of a database record.

The integration of client changes into the server requires as input a list of changed records. Since our goal is eventual consistency, only the initial and final image of changed records, as opposed to each incremental state, are needed. The initial image of a changed record refers to its state at the time of replication. It is required in order to detect update conflicts, as explained below. We are experimenting with two places for recording the initial image: the client device and the proxy. The former leads to a more scalable system, but adds complexity to client-side processing and data transmission. The latter requires maintaining on the proxy a data shadow for each client serviced. Although this demands more space on the proxy, it allows for an important optimization: only differences between two consecutive hoarding operations need to be transferred from the proxy to a client.

If a changed record does not cause update conflicts, the change is simply replayed on the server, subject to any integrity constraints enforced by the server database. If an update conflict occurs, the synchronization component will try to resolve the conflict based on data semantics. An update conflict can be either an update/update conflict or an update/delete conflict. If we associate a bit field with each record to indicate whether a record has been deleted, a deletion can conceptually be treated in the same way as an update. An update conflict occurs if and only if a record is changed on the client and the record image in the server database is not identical to the initial image of the client copy. Note that the deletion bit is physically maintained on the client, but is virtual for records on the server. When a record with the same key cannot be found in the server's database, we imagine that the server contains a record with the deleted bit on. All records actually in the server's database are considered having the deleted bit off.

The semantic information used to resolve conflicts has to be provided by either an application developer or a system administrator. Two types of information are needed. The first type is concerned with computed fields, *i.e.*, values in a database that are computable from the rest of the values. All computed fields should be identified, along with a formula for computing them. Obviously, there should be no circular definitions. Computed fields are evaluated at the end of the conflict resolution process, based on the integrated value of non-computed fields. The second type of information is integration rules. There should be an integration rule for each non-computed field, describing how to handle conflicting updates to the field. Assume that a non-computed field F had initial value f and that concurrent updates have resulted in value f_c on the client and f_s on the server. The integration rule can specify any one of the following as a resolution policy:

max (min): The maximum (minimum) of f_c and f_s becomes the integrated value of F.

client (server): The integrated value is provided by f_c (f_s).

constant k: If f_c and f_s are identical, then store that value in F. Otherwise, use constant k. This is useful for storing a null value if desired.

operational: The integrated value is set to $f_c + f_s - f$ (*i.e.*, deltas on both sides are applied to the original value).

reject: If f_c and f_s differ, reject the changes made on the client and alert the user.

program p: A program p is executed, whose parameters are f_c, f_s and f.

At the user's discretion, the synchronization component can also refresh the client-side replica. This entails updating the client-side records with the integrated state of the server's database, as well as keeping the subset of records true to its folder definition. To achieve this, a description of the data contents on the client is maintained on the proxy, which is simply folders selected and their parameter values.

3.4. User Control

The user control component provides an interface for a mobile user to intervene in the data replication on the client. A guiding principle is that the interface should require little effort from naive users and in the meantime should provide flexibility to sophisticated users who are willing to be more involved [35].

The interface allows the user to control three aspects of data replication: subsetting, targeting, and synchronization. For subsetting, the user is presented a list of folders that he may select to replicate. For each selected folder, he may either accept the default parameter value, if any, or provide his own. He can also request records that belong to a folder by scanning through the primary key values, if such a feature is allowed for the folder. In a straightforward situation, the system administrator can have each client device configured with default folders and default parameters; in this case, the user does not need to do anything about subsetting.

For targeting, the user can create, update, delete or suspend his own targeting rules. As mentioned earlier, each user-defined rule is supplemented with an extra conjunctive condition of the form "**User** $=$ xxx". Modifications made to the rules are propagated to the server at the next synchronization point, prior to any future data hoarding. Since there are already system-wide and application-specific targeting rules that perform adequately, the user is not required to provide his own rules.

Finally, for synchronization, the user can control the direction of data flow, either globally or on a per data subset basis. A data subset is determined by a unique combination of a folder and its parameter values. Hence there may be multiple subsets on a device that correspond to a single folder. There are two synchronization options:

1. Upload: Only modifications made on the client are propagated back to the server.

2. Synchronize: Uploading is followed by downloading the integrated values to the client.

Global two-way synchronization is the system default.

3.5. Rendering

The rendering component on the client device retrieves data from the XML data store and presents it in a manner appropriate for the device. Each XML document represents a data subset and contains the following information:

- Data schema: The type of each field and the record layout.

- Data values: The records that constitute the data subset.

- Semantics of exchange: Which fields are read-only, which are updatable, which can be sorted upon.

- Processing logic: Integrity constraints to be enforced.

The rendering component consists of an XML parser and an XML browser. While the former parses an XML document into a Document Object Model (DOM) [38] tree, the latter displays the data according to the characteristics of the device and controls the user interaction.

The XML browser is device-specific, but independent of which folder it manipulates. There is one generic browser for each category of client device. For a table that belongs to a folder, the browser displays a list of its records. The list may be sorted on various fields, as specified in the folder definition. The user can browse through the list or search for specific records. He can also expand individual records to get a more detailed view or to perform updates. All updates are validated using the integrity constraints. If a folder contains multiple tables, the browser also allows the user to navigate through them.

We plan on letting the application developer customize the interface for a folder if desired. An application developer will be able to describe the look and feel in XSL [40], a style sheet language for XML. If a style sheet exists, the XML browser displays the data according to the specification. Otherwise, the browser falls back on the default logic.

We mentioned earlier that the major motivation for the rendering component was application portability across various display devices. An equally important advantage is that it significantly simplifies application development. Application developers can concentrate on the semantics of data access without having to worry about the intricacies of user interface design. Thus, development time is reduced and quality is enhanced.

3.6. Network Manager

The network manager component is divided into two subcomponents: the Proxy Network Manager that resides on the proxy and the Client Network Manager on the device. As mobile clients may connect to the proxy over a variety of wireless networks (packet radio, cellular, digital cellular) as well as dial-up and LAN connections, the network manager provides an encapsulation of the communication medium to its users. The interface between the network manager and the rest of the system assumes an asynchronous communication channel. This not only allows user and system tasks to be carried out while the device is disconnected, but also favors the intermittent connectivity and/or high latency that is typical during connection.

The responsibilities of the network manager includes the following:

1. Establishing or terminating a connection as requested by the user.

2. Authenticating the user based on his password.

3. Collecting information required by the targeting component: device characteristics, link attributes and user ID.

4. Transporting user commands and data updates from the device to the proxy.

5. Transporting user-specified data subsets from the proxy to the device.

An important challenge for the network manager is how to optimize the transmissions of XML documents over a slow link. Two possible solutions are differencing and compression utilizing the well-formed structure of XML data. A detailed discussion of these techniques is beyond the scope of this paper.

4. Related Work

Although many aspects of DataX have appeared in previous systems of various contexts, we view our contribution as the generalization and engineering of existing techniques into a uniform architecture that effectively supports database access from heterogeneous types of mobile devices. In this section, we discuss work in the areas our system spans.

Several methods have been proposed to enable mobile information access. They all rely on optimistic replication control to provide high data availability and performance. The notion of disconnected operation was successfully demonstrated in Coda [33, 23], a client/server-based distributed file system. Ficus [15, 16] is another file system that effectively supports primarily disconnected mobile computers, but adopts a peer-to-peer model. Both Coda and Ficus use version vectors [29] to detect conflicts and a rule-based approach to select application-specific conflict resolvers [25, 32]. In contrast, Bayou [8, 37, 30] utilizes application-provided dependency checks and merge procedures to handle conflict detection and per-write conflict resolution respectively. For document accesses, Lotus Notes [20] allows conflicts to be automatically merged at both the document level and field level. The Rover toolkit [19] provides two mechanisms, relocatable dynamic objects and queued remote procedure calls, for building mobile-aware applications and writing proxies to enable legacy applications for mobility. IBM's eNetwork Web Express [10] combines caching and request queuing to support disconnected and asynchronous Web browsing. The need for dynamic adaptation in mobile information access has been identified by a number of projects. In particular, Odyssey [27] provides system support to enable mobile-aware applications to use data fidelity levels to control resource utilization.

Many researchers have looked into database system issues that arise in a mobile environment. Imielinski and Badrinath suggested solutions to the problems of location updates and location-dependent query processing [18]. An analysis of various static and dynamic data replication methods were presented in [17]. Barbara and Imielinski proposed an approach to invalidating the secondary replication on mobile clients [3], in which the server periodically broadcasts a report that reflects the changing database state. Krishnakumar and Jain discussed an architecture and protocols for mobile sales applications [24]. All these projects assume that the mobile clients are connected, via a wireless channel, most of the time. For disconnected operation, Badrinath and Phatak proposed clustering of data on the server based on the concept of hoard key [2]. Gray *et al.* presented a two-tier replication scheme to allow disconnected applications to request tentative transactions that are later applied to a master copy [14]. The conflict resolution mechanism in our system is based on the Data-Patch tool described in [12]. Major database vendors have just begun to offer standalone databases for hosting on mobile computers a portion of the backend database [28, 34]. However, these systems are not portable across system platforms, and they do not provide a uniform framework for adapting the data as we do.

As far as accommodating client variations is concerned, DataX has an overlap with DIANA [21, 22] and GloMop [11, 4]. DIANA advocates a form-based approach to structuring mobile applications. Applications in DIANA only need to be defined in terms of forms inputs requested from users and the application's responses to those users. Controlling user interaction in a device-specific manner is handled by a generic User Interface Logic component, thus decoupling the user interface logic from the processing logic of applications. Disconnected operation is supported via hoarded forms and application surrogates that emulate the behavior of the applications. DataX focuses on database accesses, for which a form-based interface is most natural. There is no need for separate application surrogates in DataX, as common processing logic is incorporated into the rendering component. While the synchronization process in DIANA detects incorrect decisions made by the application surrogate and gives the user an opportunity to correct the error, the synchronization in DataX detects concurrent updates and resolves conflicts based on predefined semantic information. Since DIANA predates XML, it defines its own form description language. DataX uses XML to describe the semantics of data exchange, as well as the data processing required (such as enforcing integrity constraints). Further, DataX allows customization of the user interface through XSL.

GloMop is a proxy architecture for enabling effective access to Internet content for a wide range of clients. The proxy retrieves documents on the client's behalf and performs transcoding on the fly to suit network characteristics and device capability. GloMop's emphasis is the read access of multimedia data (images and videos). Although its framework also supports transcoding, DataX is mainly concerned with enterprise data, which is mostly text. DataX prepares those data for mobile access via subsetting and filtering. Since updates are part of life, DataX also provides a mechanism to integrate divergent copies. Finally, data adaptation policies in GloMop are specified by client applications through a special API. For legacy applications, a client-side agent is required to communicate with the proxy. In comparison, a rule-based interface for defining adaptation policies, like the one used in DataX, not only makes it easy to support legacy applications, but also allows for customized adaptation.

5. Summary and Conclusions

DataX is a proxy-based architecture that extends the reach of enterprise data to the users of thin mobile clients. It supports disconnected operation by replicating necessary data on the client device using a weak consistency criterion. Data replication can be adapted to device characteristics, link attributes and user preferences. Divergent replicas on the client and on the server are integrated through an automated synchronization process. The proxy can interface to any ODBC-compliant enterprise database, making it independent of server platforms. The proxy exchanges data with the client device in XML format. A per-device renderer displays the data in a form layout and controls the user's manipulation. DataX provides an interface for application developers to define the data subsets, and policies for targeting and integration. The mechanisms that implement these policies as well as other common functions are built into the infrastructure.

Since the bulk of the work is performed on the proxy, the client software can be extremely lightweight and the end-user has to do very little administration. This is useful for resource-limited thin clients and casual users. By decoupling user interface logic from application logic and allowing device-specific customization, it is simple to provide application portability across heterogeneous client devices and to introduce new device types. The application developer's job is also made easier, as he can concentrate on the semantics of data access without having to worry about the underlying mechanisms and interface issues. This reduces the time to deploy end-to-end solutions and can be a major competitive advantage for organizations.

References

[1] R. Alonso, D. Barbara, and H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM Transactions on Computer Systems*, 15(3):359–384, September 1990.

[2] B. R. Badrinath and S. Phatak. Database Server Organization for Handling Mobile Clients. Technical Report DCS-TR-324, Department of Computer Science, Rutgers University, 1997.

[3] D. Barbara-Milla and T. Imielinski. Sleepers and Workaholics: Caching Strategies in Mobile Environments. *ACM SIGMOD Record*, 23(2), May 1994.

[4] E. A. Brewer and R. H. Katz *et al.* A Network Architecture for Heterogeneous Mobile Computing. *IEEE Personal Communications: Special Issue on Mobile Access to Web Resources*, 5(5):8–24, October 1998.

[5] M. Butrico, H. Chang, A. Cocchi, N. Cohen, D. Shea, and S. Smith. Gold Rush: Mobile Transaction Middleware with Java-Object Replication. In *Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems*, Portland, Oregon, June 1997.

[6] J. Creamer, M. O. Stegman, and R. P. Signore. *The ODBC Solution: Open Database Connectivity in Distributed Environments*. McGraw-Hill, 1995.

[7] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in Partitioned Networks. *Computing Surveys*, 17(3):341–370, September 1985.

[8] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing Among Mobile Users. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[9] A. Elmagarmid, J. Jing, and T. Furukawa. Wireless Client/Server Computing for Personal Information Services and Applications. *ACM SIGMOD Record*, 24(4):16–21, 1995.

[10] R. Floyd, B. Housel, and C. Tait. Mobile Web Access Using eNetwork Web Express. *IEEE Personal Communications: Special Issue on Mobile Access to Web Resources*, 5(5):47–52, October 1998.

[11] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 160–170, Cambridge, Massachusetts, October 1996.

[12] H. Garcia-Molina, T. Allan, B. Blaustein, R. M. Chilenskas, and D. R. Ries. Data-Patch: Integrating Inconsistent Copies of a Database after Partition. In *Proceedings of the 3rd Symposium on Reliability in Distributed Software and Database Systems*, pages 38–44. IEEE, October 1983.

[13] A. Goel. View Consistency for Optimistic Replication. Master's thesis, University of California, Los Angeles, 1996. Available as UCLA Technical Report CSD-960011.

[14] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proceedings of 1996 ACM SIGMOD Conference*, pages 173–182, June 1996.

[15] R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page Jr., G. J. Popek, and D. Rothmeier. Implementation of the Ficus Replicated File System. In *Proceedings of 1990 USENIX Summer Conference*, pages 63–71, June 1990.

[16] J. S. Heidemann, T. W. Page, R. G. Guy, and G. J. Popek. Primarily Disconnected Operation: Experiences with Ficus. In *Proceedings of 2nd Workshop on the Management of Replicated Data*, pages 2–5, Monterey, CA, November 1992.

[17] Y. Huang, P. Sistla, and O. Wolfson. Data Replication for Mobile Computers. In *Proceedings of 1994 ACM SIGMOD Conference*, pages 13–24, 1994.

[18] T. Imielinski and B. R. Badrinath. Querying in Highly Mobile Distributed Environments. In *Proceedings of 18th Conference on Very Large Data Bases*, pages 41–52, August 1992.

[19] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers: Special Issue on Mobile Computing*, 46(3), March 1997.

[20] L. Kalwell Jr., S. Beckhardt, T. Halvorsen, R. Ozzie, and I. Grief. Replicated Document Management in a Group Communication System. In D. Marca and G. Bock, editors, *Groupware: Software for Computer-Supported Cooperative Work*, pages 226–235. IEEE Computer Society Press, 1992.

[21] A. M. Keller, T. Ahamad, M. Clary, O. Densmore, S. Gadol, W. Huang, R. Razavi, and R. Pang. The DIANA Approach to Mobile Computing. In T. Imielinski and H. F. Korth, editors, *Mobile Computing*, pages 651–679. Kluwer Academic Press, 1995.

[22] A. M. Keller, O. Densmore, W. Huang, and R. Razavi. Zippering: Managing Intermittent Connectivity in DIANA. *ACM MONET (Mobile Networks and Applications)*, 2(4), 1997.

[23] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[24] N. Krishnakumar and R. Jain. Protocols for Maintaining Inventory Databases and User Service Profiles in Mobile Sales Applications. In *Proceedings of the Mobidata Workshop*, October 1994.

[25] P. Kumar. Flexible and Safe Resolution of File Conflicts. In *Proceedings of 1995 USENIX Winter Technical Conference*, New Orleans, LA, January 1995.

[26] H. Lei. *Uncovering and Exploiting the Intrinsic Correlations between File References*. PhD thesis, Columbia University, 1998.

[27] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of 16th ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, October 1997.

[28] Oracle Corporation. Oracle Lite Reviewers' Guide, 1998. http://www.oracle.com/mobile/olite/html/ol_review.pdf.

[29] D. S. Parker Jr., G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Transactions on Software Engineering*, SE-9(3):240–247, May 1983.

[30] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proceedings of 16th ACM Symposium on Operating Systems Principles*, Saint Malo, France, October 1997.

[31] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.

[32] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek. Resolving File Conflicts in the Ficus File System. In *Proceedings of 1994 USENIX Summer Conference*, June 1994.

[33] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[34] Sybase, Inc. SQL Anywhere Studio - A Guide for Evaluation and Review, 1998. http://www.sybase.com/products/-anywhere/sql_reviewers_guide.pdf.

[35] C. Tait, H. Lei, S. Acharya, and H. Chang. Intelligent File Hoarding for Mobile Computers. In *Proceedings of 1st International Conference on Mobile Computing and Networking*, pages 119–125. ACM, November 1995.

[36] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session Guarantees for Weakly Consistent Replicated Data. In *Proceedings of 3rd International Conference on Parallel and Distributed Inormation Systems*, Austin, TX, September 1994.

[37] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of 15th ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, December 1995.

[38] The World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification, 1998. http://www.w3c.org/-TR/1998/REC-DOM-Level-1-19981001/.

[39] The World Wide Web Consortium. Extensible Markup Language (XML) 1.0, 1998. http://www.w3c.org/TR/1998/-REC-xml-19980210.html.

[40] The World Wide Web Consortium. Extensible Stylesheet Language (XSL) Version 1.0 Working Draft, 1998. http://www.w3c.org/TR/1998/WD-xsl-19980818.html.

[41] O. Wolfson, P. Sistla, S. Dao, K. Narayanan, and R. Raj. View Maintenance in Mobile Computing. *ACM SIGMOD Record*, 24(4):22–27, 1995.