A Three-tier Architecture for Ubiquitous Data Access

Sumi Helal, Joachim Hammer, Jinsuo Zhang, and Abhinav Khushraj

Computer ad Information Science and Engineering Department
University of Florida, Gainesville, FL 32611
{helal,jhammer,jizhang,abhinav}@cise.ufl.edu

Abstract

In this paper we present a three-tier architecture of a middleware that addresses challenges facing accessibility, availability, and consistency of data in mobile environments. The architecture supports the automatic hoarding of data from multiple, heterogeneous sources into possibly a variety of different mobile devices. The middle tier eliminates the manual and tedious synchronization currently done by the user, between the mobile device and the fixed network on the one hand, and between multiple mobile devices owned by the user on the other hand. This middleware enables the automation of synchronization tasks in both connected mode (following disconnection) and weakly connected mode, where only intelligent and effective synchronization can be used in the presence of a low-bandwidth network. We present the three-tier architecture and describe implementation and experimental work status of an incremental hoarding mechanism based on the Coda file system.

1. Introduction

In today's networked computing environment, users demand constant availability of data and information which is typically stored on their workstations, corporate file servers, and other external sources such as the WWW. An increasing population of mobile users is demanding the same when only limited network bandwidth is available, or even when network access is not available. Moreover, given the growing popularity of portables and personal digital assistants (PDA), mobile users are requiring access to the data regardless of the form-factor or rendering capabilities of the mobile device they choose to use.

We identify three broad and interdependent challenges imposed by mobility:

1. Any time, anywhere access to data, regardless of whether the user is connected, weakly connected via a high latency, low bandwidth network, or completely disconnected;
2. Device-independent access to data, where the user is allowed to use and switch among different portables and PDAs, even while mobile;
3. Support for mobile access to heterogeneous data sources such as files belonging to different file systems and/or resource managers.

In this paper we present a three-tier architecture that addresses the aforementioned challenges, and that supports the automatic hoarding of data from multiple, heterogeneous sources into possibly a variety of different mobile devices. The middle tier eliminates the manual and tedious synchronization currently done by the user, between the mobile device and the fixed network on the one hand, and between multiple mobile devices owned by the user on the other hand. This middleware enables the automation of synchronization tasks in both connected mode (following disconnection) and weakly connected mode, where only intelligent and effective synchronization can be used in the presence of a low-bandwidth network. Currently, there are no such architectures developed, neither within research projects nor as commercial products.

We build upon existing results and systems to develop architecture and algorithms to make smart hoarding and synchronization in mobile environments a reality. We see the realization of such a framework as an important and necessary step towards making mobile computing a viable practice for a broad audience; a step that some day may lead us to the realization of ubiquitous computing. Users should

be allowed to switch between any mobile device to connect to the fixed network and carry the necessary data with them, without having to worry about hoarding, synchronization, and other device-specific low-level burdens.

In manual hoarding a mobile user has to decide the data sets he wants to carry with him before going mobile, manually copy and synchronize in reverse direction upon return, decide whether to store the new set of files and directories on the fixed network or his mobile computer. These tasks are further complicated if a mobile user has more than one mobile device. They are even more complicated if the data comes from different heterogeneous sources (e.g., different file systems). Manual hoarding and synchronization is a time consuming process and prone to errors as everyone can attest who has forgotten to download a critical report before leaving the office, or attempted to update a file that has changed since it was last read.

In our approach, if the user wants to become mobile, he first connects to his "*mobility account"* using a mobile device of his choice. Immediately, the user's working set, which is maintained persistently in a warehouse, is retrieved, and an incremental hoarding process is initiated to update the contents of the mobile device. The hoarding process remains active so long as the user is connected. Upon disconnection, a sensor located on the mobile device will track the usage of the data items residing on the mobile device and also record any requests for data not currently present ("access faults"). Upon return, the access faults will be used to update and further improve the contents of the working set in the warehouse and thus the contents of the mobile device. In addition, synchronization of the working set in the warehouse with the corresponding data files on the fixed network and with the cache on the mobile device is conducted; in case of a weak connection during the roaming period, synchronization as well as updates to the mobile working set is possible. In this way, all data management activities that involve the copying and synchronization of files are carried out automatically. From a user's perspective, the only manual interventions required are in the beginning, when seeding the working set with important files and during synchronization in order to resolve certain conflicts that cannot be handled by our synchronization algorithms.

In Section 2 we discuss a three-tiered data hierarchy that will enable data abstraction in the middle layer. Section 3 describes the architecture in detail. In Section 4 we describe related research and conclude the paper in Section 5

2. Three-tiered Data Hierarchy

So far we have motivated the need for more advanced management techniques for mobile data access and update. Current research related to mobile data management is lacking support and integration of ubiquitous access (from any mobile device), heterogeneous access (to any source of data be it structured, unstructured, or semi-structured), and consistent access and update (in all connection modes: connected, weakly connected or disconnected). In the following, we introduce our new, three-tiered architecture and framework to mobile data management.

The key to our approach to automating mobile data management is the abstraction of user data (that is most likely to be used during roaming or disconnection) into a repository, independent of the computing devices that are used and the sources where the data originates. To this end, we have developed an architecture based on concepts developed in database systems, specifically in data warehousing. In our approach, the existing two-tier data hierarchy of mobile computing (mobile nodes and fixed network) is extended to include a middle tier consisting of a data warehouse to produce the three-tier data hierarchy shown in Figure 1. The three tiers are:

- The *data sources,* which include but are not limited to file systems, database servers, workflow engines, e-mail servers, Web servers, etc.
- The *working set* (one per user), which are de-coupled from mobile devices.

- The *mobile caches*, each of it contain copies of a subset of the user's working set.

By introducing a new middle tier we can separate the mobile nodes from the data sources, shielding each from the changes that have affected the others. The middle tier, which is a data warehouse, acts as a mobility-aware persistent store. When the user is connected, it accumulates the user's working set. When the user is roaming, it collects updates affecting the disconnected users and keeps the users' working sets up-to-date. When the mobile user returns, it synchronizes the collected updates in the warehouse with the contents of the mobile cache (tier 3), as well as updates in the mobile cache with the contents of the sources.

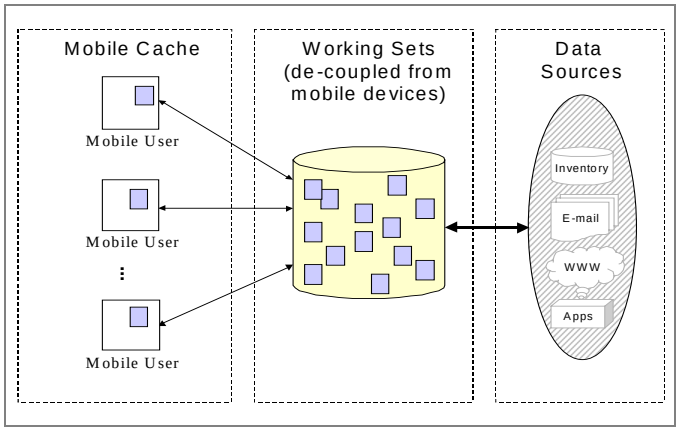


Figure 1: Three-tiered data hierarchy for mobile data management.

3. System Architecture and Components

We have designed an event-driven architecture, in which the behavior of the mobile environment manager is determined by the actions specified in rules (e.g., using the event-condition-action paradigm). This approach fits well with the event-driven nature of a mobile computing environment as well as with some existing approaches for maintaining the contents of data warehouses.

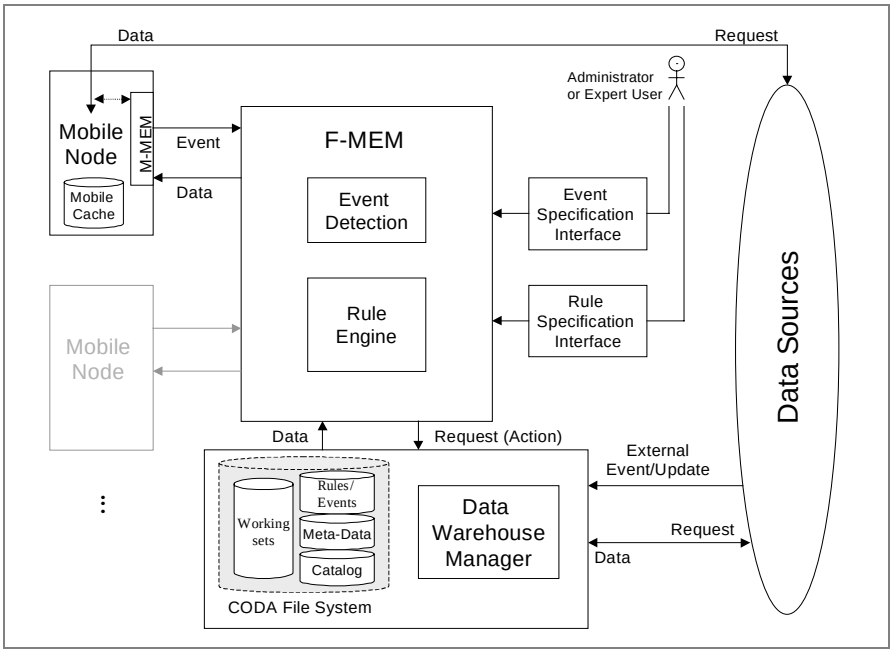


Figure 2: Architecture of the mobile environment manager.

Figure 2 shows the designed architecture. The heart of the architecture is the mobile environment manager or MEM, which is divided into a fixed network component F-MEM and a mobile component, M-MEM.

3.1. Mobile Environment Managers (MEM)

On the left side of Figure 2 are the mobile users (nodes). Each node has the mobile part of the mobile environment manager (M-MEM). The main tasks of M-MEM can be described as follows:

- When the mobile device is connected, "sense" all data requests by the mobile user (for local as well as remote data) and generate the appropriate events to F-MEM. Based on these events, F-MEM may decide to perform hoarding or synchronization tasks, or may only re-evaluate the user working set. It is worth pointing out that when in docked mode, the user of the mobile device will access the source data on the fixed network directly (and not a copy that happens to be hoarded in the mobile cache), as indicated by the solid line connecting the mobile node to the data sources.
- When the mobile device is in roaming or disconnected mode, continue to sense and store the access patterns and accompanying events for later download to the F-MEM. In case of weak connection, prioritize requests for missing data and attempt to upload as much as possible based on this priority.

The basic functionality of the F-MEM is to accept registration of events, detect events, and trigger the appropriate event-condition-action (ECA) rules. The main tasks of the F-MEM are to compute the working set, instruct warehouse manager to copy data items from the sources, coordinate hoarding activities, detect disconnection and reconnection, detect conflicts and synchronize between the mobile nodes and the warehouse.

3.2. The Warehouse

Persistence is provided by the data warehouse component shown underneath F-MEM. We need persistence for the following four groups of data items: (1) rule and event specifications, (2) user data, (3) meta-data, and (4) data catalog as well as other operational data.

The rule and event specifications control the behavior of the hoarding and synchronization to be performed by MEM. The data catalogs refer to the working set of each user. This is automatically computed by the F-MEM.

The meta-data includes accounting information and profiles for each mobile user. This includes his usage patterns, his current connect status, as well as the meta-data describing his current working set.

The functionality of the warehouse storing the working sets includes:

- monitor information sources and generate update notifications in case the contents have changed (e.g., based on work done by [4]);
- maintain specified consistency levels between source data and the copies in the warehouse using the update notifications mentioned above (e.g., based on work on incremental warehouse maintenance by [12]);
- store a wide spectrum of data from heterogeneous data sources

The warehouse is designed as a Coda file system, which it uses to store the user data. The warehouse manager incrementally hoards the user data from the different sources into the warehouse. The user data is stored within the warehouse as file volumes.

The rest of the data (rules, meta deta, catalogs) inside the warehouse is maintained in a relational database system which is also stored on the same Coda server. The rules and the events are specified by human experts and are communicated to the warehouse by the F-MEM. The meta-data is also stored in the relational system. The working set for every user is evaluated using the meta-deta and its content information is maintained by the data catalogs inside the warehouse.

At the time of hoarding the cache of a mobile user, the working set is evaluated by the F-MEM and the corresponding file volumes are hoarded into the mobile user directly from Coda. This is done using an efficient technique that has been implemented, as explained in the next section.

The warehouse also maintains per-object consistency criteria for different connection levels (disconnected, connected, weakly connected modes). These are triggered by the appropriate rules and events maintained by the warehouse.

The motivation behind using coda for storing data at the file granularity is that by doing it this way we can leverage on the basic hoarding mechanism supported by Coda together with our incremental hoarding based on version control. In addition, we can leverage Coda's re-integration mechanisms into our per-object variable consistency approach both upon reconnection and during periods of weak connections.

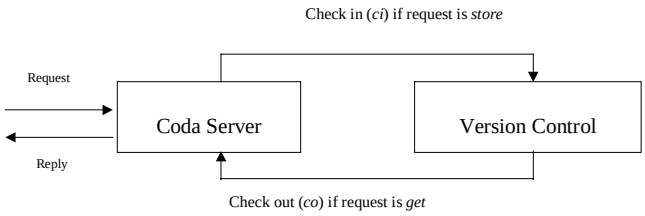
3.3. Incremental Hoarding

Hoarding is a necessary step in preparation for anticipated disconnection. Disconnecting from a weak connection mode (current wireless data services such as GSM, iDEN, etc.) makes hoarding procedures heavier under the limited bandwidth constraint. We observed that the Working Set does not change abruptly between successive hoarding sessions. A file determined to belong to the Working Set is often already hoarded, or a slightly different version of it. The size difference between the new and the past hoarded versions is often minor. In incremental hoarding, only delta files are transferred, which reduce the bandwidth requirement for hoarding -- an important objective for hoarding under weak connection mode. For example, in the Coda system, if one hoarded file at the client side becomes stale, Venus (Client side cache manager) will simply discard it and fetch the entire up-to-date version of this file to the mobile device. In a high speed LAN environment, this is not a problem. In the mobile environment, however, the client side is often a laptop and the connection between the client and the server is often through a dial-up (wireless or wire-line) modem with limited bandwidth. Frequently fetching an entire file from the server is an expensive process, especially if the modifications made to the file is relatively minor.

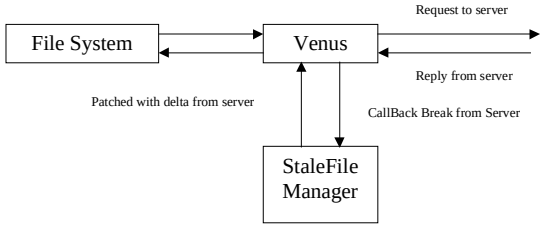
We have implemented simple incremental hoarding based on the Coda file system [6, 9, 10] and using the Revision Control System (RCS). The implementation is two parts: a server side and a client side, as shown in Figure 3.

At the server side, there are two scenarios for file update: connection-mode re-storing and reintegration. In the former, a user directly opens a Coda file, modifies it and saves it. Reintegration happens in disconnection mode or weak connection while write propagation is disabled. Then once reconnected, Venus automatically propagates all the modifications to the server. In both cases, the server should store the newer version file into the corresponding RCS file. In RCS terms, it is checked in. When new version is saved, then RCS command *ci* is used. The RCS file name of a file is the concatenation of this file with ",*v*" string. In Coda, when the new version file is stored, a new file is created and contents are saved, old file is deleted automatically. Therefore, the corresponding RCS file should also be renamed accordingly.

When Venus wants to fetch a file, it should provide the *version number* of the stale file along with all other information such as *id* of fetched file. In the original Coda system, Venus always fetches the whole file, so it is unnecessary to provide version number. In this implementation, a file version number is provided by an RPC call. If the version number provided is 0, it means that Venus actually has no stale version of the current file, then the server will retrieve (check out, in RCS terms -- the command for RCS is *co*) the up-to-date version of the file. If the version provided by Venus is greater than 0, then Vice will get the difference between provided version and the up-to-date version by using RCS tool *rcsdiff*. The difference is transferred to Venus using the FTP protocol. The information set by the server is still the information about the up-to-date file, not the difference file.



(a) Server side support



(b) Client side support

Figure 3. Incremental Hoarding in Coda

When Venus first fetches one file, it is exactly the same as before (before modifying Coda). When one file is fetched, it is cached and one Callback is requested from the server. If someone else updates this file, the server will break the Callback (the cache file is invalidated by the server). Without incremental hoarding, Venus will simply discard the cached file. In order to implement incremental hoarding, we implemented a *StaleFileManager*.

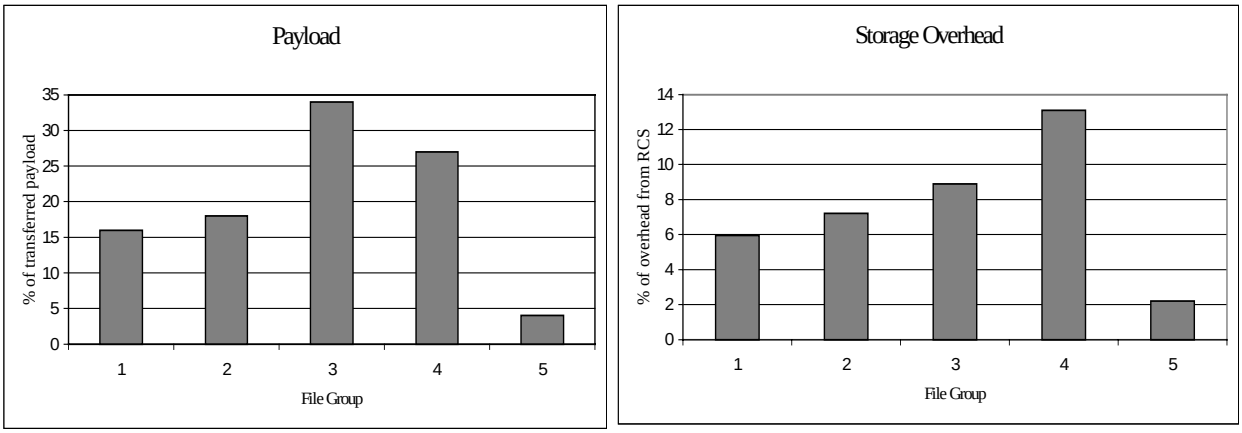
When one cached file is Callback broken by the server, Venus will discard this stale state file from its own cache library. Then the StaleFileManager takes over. From the viewpoint of Venus, this cached file is deleted, but it is actually saved by the StaleFileManager.

StaleFileManager provides Venus with information regarding which stale files and version numbers are kept. Then Venus could provide this information to the server when doing incremental hoarding. When a difference file is transferred from the server, StaleFileManager provides the stale file so that Venus could combine the two files to obtain an up-to-date version. Upon renaming and deleting files in Coda, the StaleFileManager follow by updating the corresponding information in its own library.

In order to evaluate the performance gain and overhead from incremental hoarding, we conducted simple experiments to compare incremental hoarding and original Coda hoarding. A total of 100 files are randomly selected from Linux kernel source codes provided by RedHat 6.0 and RedHat 6.1. (The actual kernel code versions are 2.2.5 and 2.2.12). Every 20 files are grouped into one group, so altogether 5 groups are made. At first, RedHat 6.0 is hoarded into one client, then another client upgrades the code in the server into RedHat 6.1. Then the first client needs to invalidate the original files and fetch a newer version. Both transfer payload and storage overhead are measured and compared.

Figure 4(a) shows the comparison for transferred payload. The original hoarding is used as base, and the columns showed here are the percentages of transferred payload for incremental hoarding relative to base. From the figure, it is obvious that incremental hoarding only spent a small percentage of network load.

Figure 4(b) shows the storage overhead for incremental hoarding. The columns show the extra storage requirement for incremental hoarding relative to the original hoarding strategy. From the figure, we see that the overhead is not significant. Furthermore, since Coda server maintains multiple versions of the file, the client could be serviced with multiple versions. This opens the door to additional consistency mechanisms in Coda (versioning-based concurrency control).



(a) (b)

Figure 4. Performance using Incremental Hoarding

4. Related Research

- *Mobile Data Management.* Recent research in mobile data management (a survey can be found in [5]) has been focused on transactional and non-transactional access and delivery of data between servers and mobile clients. The Coda file system [6, 9, 10] provides an application-transparent file system for mobile clients by using file hoarding and optimistic concurrency control. Unlike traditional transactions, it does not guarantee failure atomicity and only conditionally guarantees permanence. Automatic mechanisms for conflict resolution are provided for directories and files through the proxy and the file server. Hoarding is based on user-provided, prioritized list of files. The SEER system [7] also provides interesting hoarding algorithms based on the Coda file system basic hoarding mechanism. The DataX project at IBM Watson Research Center [8] uses a related middleware architecture that focuses on data rendering over a variety of heterogeneous devices. DataX also addresses issues of consistency adaptation based on the type of connection and user profiles.
- *Data Warehousing.* An important requirement for our approach to mobile data management is to be able to store and maintain the working sets of mobile users in light of frequent changes. Since the warehouse only maintains a copy of the data, the actual changes occur elsewhere and must be propagated to the warehouse as efficiently as possible. Many incremental view maintenance algorithms have been developed for centralized database systems, for example [1,2], and a good overview of materialized views and their maintenance can be found in [3]. [11] showed how to provide consistency in a single-source environment, [12] extend these techniques and provide a solution for more general case of multiple sources and transactions that may span sources.

5. Conclusions

In this paper, we have presented a data warehouse based three-tiered architecture for mobile data management. Our work is based on current research in mobile computing and data warehousing. We address key issues where current research is either lacking or is needy of integration. In particular, our approach provides: support for ubiquitous data access (where hoarding and synchronization tasks are mobile device independent); hoarding and consistency maintenance from heterogeneous data sources (using a data warehouse as a middle layer); flexible synchronization through programmable and conditional consistency specification of mobile data items. We presented implementation status and preliminary experiments, which at this stage, are focused on Coda-based efficient and incremental hoarding algorithms.

The investigation is important given the massive growth of the portable computer and wireless data networking industries. More users and businessmen than ever before own portable computers and other mobile devices such as hand-held and palm computers. Moreover, what so called Wireless IP services are being offered in the U.S. through fierce competitions. The stage is set for mobile computing to flourish. Our research is a step forward towards providing the necessary support for the development of highly usable mobile data management systems.

6. References

[1] S. Ceri and J. Widom, "Deriving Production Rules for Incremental View Maintenance," in *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, Barcelona, Spain, pp. 577-589, 1991.

[2] A. Gupta, I. Mumick, and V. Subrahmaninan, "Maintaining Views Incrementally," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, D.C., pp. 157-166, 1993.

[3] A. Gupta and I. S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications," *Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, **18**:2, pp. 3-18, 1995.

[4] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo, "Extracting Semistructured Information from the Web," in *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, 1997.

[5] J. Jing, A. Helal, and A. Elmagarmid, "Client-Server Computing in Mobile Environments," *ACM Computing Surveys*, 1999.

[6] J. Kestler and M. Satyanarayanan, "Disconnected Operation in the CODA File System," *ACM Transactions on Computer Systems*, **10**:1, pp. 3-25, 1992.

[7] G. Kuenning and G. Popek, "Automated Hoarding for Mobile Computers," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'97)*, St. Malo, France, pp. 264-275, 1997.

[8] H.Lei, M. Blount, and C. Tait. "DataX: An Approach to Ubiquitous Database Access", in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, 1999.

[9] B. Noble, M.Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker, "Agile Application-Aware Adaptation for Mobility," in *Proceedings of the Sixteenth ACM Symposium on Operating Systems*.

[10] M. Satyanarayanan, J. Kistler, P. Kumar, M. E. Okasaki, E. H. Seigel, and D. C. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, **39**:4, pp. 447-459, 1990.

[11] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, "View Maintenance in a Warehousing Environment," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, **24**:2, pp. 316-27, 1995.

[12] Y. Zhuge, H. Garcia-Molina, and J. Wiener, "Consistency Algorithms for Multi-Source Warehouse View Maintenance," *Journal of Distributed and Parallel Databases*, **6**:1, pp. 7-40, 1997.