Mobile Data Access:
Disconnected and Partially Connected Approaches

Rich Kong, Stephen Lau, Daniel New
Department of Computer Science
University of California, San Diego
La Jolla, CA 92193
{rkong, slau, dnew}@cs.ucsd.edu

March 5, 2001

Abstract
The pervasiveness of mobile devices has led to increased research in area of mobile data access. Mobile devices have created new requirements for data accessibility. They are used across a variety of networks with different bandwidths and qualities of service. At times, networks may be inaccessible and data must be accessed via a local copy. In any of these situations, mobile devices must adequately function to be considered useful. This paper describes the various techniques and current research in solving this mobile data access problem. Furthermore, these methods are synergetic and are best used collectively to provide an elegant solution to mobile data access.

1. Introduction

Recently, the computer industry has experienced an explosion in number and availability of mobile computing devices. These devices include but are not limited to laptops, palm-sized PCs, cellular phones, MP3 players, and digital cameras. They are playing an increasingly important role in our daily lives by creating a more mobile society. We are no longer confined to one location from where we can download our stock reports, listen to MP3s, and wait for phone calls. There is no doubt that the number and quality of devices will only dramatically increase in the near future.

Mobile devices exhibit major differences from their desktop counterparts. Typically, they only allow for a subset of operations that the desktop is capable of. They are designed with mobility in mind and therefore must be small and lightweight. Being small and lightweight does not come for free. The developers must make cost and design tradeoffs in order to fit a computer into such a small space. Often, they sacrifice performance leaving limited resources, such as minimal storage, limited connectivity, and low battery life. With these limitations, what makes mobile devices so desirable?

The attractiveness of mobile devices is their ability to go anywhere. Users want to be able to take their computer wherever they go and still access the same applications and documents. As users travel with their mobile devices, they desire a relatively similar user experience as if fully connected on their home network. The mobile device may travel across a variety of networks, each with its own bandwidth limitations and quality of service. Some networks may be low-bandwidth with high-delay while others are high-bandwidth with low-delay. The mobile device should function adequately in either of these partially

connected situations as well as those that lie in between. Sometimes a network may not even be present. In these times of disconnected operation, data should still be accessible and any modifications should be effectively saved and updated upon reconnection. All of these situations are effectively categorized under mobile data access.

Researchers have proposed a variety of solutions to deal with mobile data access. Those solutions range from the application-transparent caching to the application-aware adaptation. This paper analyzes these solutions and describes the advantages and disadvantages of each. Furthermore, we show that the solutions are synergistic and can be used collectively to provide an elegant solution to mobile data access.

2. The Approaches to Mobile Data Access

This section describes and evaluates the various approaches for disconnected and partially connected operation, as well as those for update propagation and conflict resolution. Disconnected approaches include predictive file caching and hoarding to anticipate user demands. Partially connected approaches couple the operating system and applications to choose the essential data to access, as well as allow for applications to adapt in their changing environments. Update propagation and conflict resolution deals with how modified files are updated back to servers and how modification conflicts are handled.

2.1. Preparing for Disconnection

The majority of mobile devices share data with a more powerful personal computer or server. Typically, this consists of a networked environment with laptop users. While connected, users have access to all of the network resources; however, special provisions must be made to provide the mobile users with the resources they need in the event of a disconnection. A similar problem is involved in a standard network of workstations. A network failure may cause resources to become inaccessible. In both of these cases, the file system on both the client and server has to be prepared for the disconnection period, whether it is voluntary (such as users unplugging their laptop from the network) or involuntary (such as network failure). The main consideration when preparing for disconnection is determining what files the users need.

The two main methods of determining what files users need are *predictive-caching* [3] (also known as *pre-caching or pre-fetching*), and *hoarding* [3]. Mobile devices usually have less memory (both RAM, and hard storage), so it is not feasible to have a duplicate copy of every file on their desktop computer or server. Therefore they need a *working set* [4] of files, which is a subset of their regular desktop files. These working sets are the mobile device's prediction of the files users may need while away from their desktop, or disconnected from their network.

2.1.1. Predictive Caching and Working Sets

While a working set is not as much an issue on notebook devices, it is still very much an issue on smaller embedded devices or PDAs where memory is scarce. The mobile

devices must be able to act in concert with the servers (a term which we use to represent either a network server, or a desktop "master" PC) to analyze the past usage of the users, and predict the files they may need in the future. Working sets are determined by the past computing usage of the users [4] and are usually traces of what files they have been working on within a certain time period. The primary difficulty in determining a working set is in setting a "cut-off" point, i.e.: looking at the usage-history and determining at what point the user started a new working set. This can be done as easily as noticing that the user went from drawing graphics in an art program to typing up a budget in a spreadsheet program. Often though, as projects become more heterogeneous and multimedia projects become more prevalent, this can be a difficult task. One approach to determining what files are needed was done by Lei and Duchamp [6]. In their approach, they attempted to design a system by which the file system would analyze the computing activities of a user and selectively pre-fetch files. While their work was based around partial (or low-bandwidth) connectivity, it is still certainly relevant to disconnected operation.

The Lei and Duchamp solution is based on building *access trees*, which are rooted by each executing application. As the application accesses files, the system adds leaves to the tree. As the application forks itself, or spawns other executable processes, the system roots sub-trees as nodes in the parent tree. All access trees are saved with the application they were rooted by. Any time the application launches, the system loads its associated access trees into virtual memory, and starts building a current working *pattern tree*. The pattern tree follows the same rules as for building the access tree, since they are the same construct. Every time a node or leaf is added to the tree, the pattern tree is compared to all previous access trees. If a compatibility with previous access trees exists, then the file-system pre-fetches the files that the matching access tree claims the application will use next.

An attention shift occurs when users radically change their working sets, such as switching from working on one project to another. [4] Depending on the distribution of files the system has managed to predictively cache, such a change can be a large problem. If the mobile device has a sufficiently large storage space, then the system will hopefully have cached the necessary files onto the device despite a low prediction rate. However, nothing prevents users from opening files they may not have used in many years and are therefore not cached. In this case, a possible solution would be to have the device connect back to the server and download the required files.

2.1.2. Hoarding

An alternative to running an algorithm to predict what files a user may need is to have the user explicitly state them. Coda uses this approach where the client specifies a *hoard database* [3], which is essentially a list of, files which the user thinks will be needed in the event of a disconnection. The system may also have its own specified files, which are always necessary while disconnected, i.e. certain essential operating system binaries such as the X-windows server, or key libraries that are needed by applications.

Hoarding has the benefit of allowing the user to specify the needed files, rather than having the system guess, as in predictive caching. This way the system caches only what the users state they need. For instance, in pre-caching, the system designer obviously wants to have the system err on the side of caution, so it is most likely that the client will cache more than is needed. This cuts down on client free-space during the disconnection period.

The obvious downside to hoarding lies in the fact that its most effective use is in preparing for a known disconnection. Hoarding can be used periodically to bring files off the server, onto the client; however, it is possible for a disconnection to occur without prior knowledge (network failure), in which case the client may not have hoarded all the files the user needed. The only way to prevent this is to decrease the period between hoard updates. However, making the period too small will force a significant increase in CPU, disk, and network usage.

2.2. Partially Connected Application-Aware Computing

A partial connection is a connection that does not have the high bandwidth or quality of service of a full connection, yet is not completely disconnected from a network. These partial connections differ from fully connected and disconnected connections in that they have varying degrees of connectivity, bandwidth, and quality.

Under partial connections, given the limited bandwidth, it is not feasible to download every file that a user requires. A solution is application adaptation to provide consistent service. In partially connected application-aware methods of mobile data access, the operating system either provides applications a special API to specify the essential data to download or notifies applications of changes in the environment, allowing them to adapt. These two different approaches are best characterized by the Prayer file system (PFS), and Odyssey.

2.2.1. Partial Connection in the Prayer File System (PFS)

The PFS [1] differs from conventional file-systems in that in addition to supporting both fully connected and disconnected connections, it also supports partial connections. In partial connections, PFS allows applications to specify the essential, individual parts of a file, or the records, that require consistency with the server. Specifying only the essential parts allows for minimal download of data. The other nonessential parts are accessed from cached copies. This is a new level of consistency since most other protocols are at the file level. PFS lets the application specify the individual record. For applications that do not need such cache consistency mechanisms, they can exist without modification allowing the file system
to work transparently.

The main advantages of the system are the partial-file caching consistency allowing for effective processing in different network environments and also the transparent support for applications that do not require any changes. With only essential data downloaded, this solution helps keep the impact of network bottlenecks and quality of service minimal.

Furthermore, applications are given more power as they determine what data is downloaded. The problem is that this still only allows for one more level of connectivity. An application cannot tell the difference between slow, semi-slow, or medium bandwidth connections. It can only tell the operating system what data is important to maintain consistent. This problem stems from a deeper problem: communications is only one-way, flowing from the application to the operating system. No conventions exist allowing the operating system to communicate to the application.

2.2.2. The Fetch-Only Approach

Another partially connected file system [2] was used with Mach 2.5. The file system performs operations locally and logs them. The logs are then replayed to the server. The cache manager handles the management supporting three modes: connected, disconnected, and fetch-only. The "fetch-only" mode allows for caching without the full replay level of "connected" but without the failure level of "disconnected" mode. Updates are done to a local log, which is then replayed to the server by a background daemon. When the Andrew benchmark was performed on the system, it revealed that on a slow link, the partial connection/fetch-only option improved the performance significantly than the fully connected mode. It ran about 10% slower than on a fully connected Ethernet and about 4 times faster than fully connected on a slow link.

2.2.3. Agile Application-Aware Adaptation with Odyssey

Odyssey is a system that attempts to facilitate *application adaptation* [9] to cope with partial connections. By allowing applications to sense and adapt to changing network quality they can provide consistent service. Odyssey monitors resource availability and communicates changes to interested applications while enforcing distribution decisions. It explicitly defines resource types in order to better measure access availability. This can include video, audio, images, etc. *Wardens* [9] manage and monitor a specific type. Applications request a resource and define what tolerance in quality they will accept. The system notifies them when the wardens detect resource availability strays out of bounds. Odyssey performs notifications via callbacks, which allows rapid communication of such changes. Such agility is important since large and erratic changes in network quality require rapid adaptation to provide adequate service. This benefit is the ability to provide support near real-time access to data. Furthermore, applications can adapt to changing environments perhaps adjusting what data they require to perform satisfactorily. The disadvantage of this close coupling between Odyssey and applications is a lack of transparency, which limits portability and legacy support.

2.3. Update Propagation and Conflict Resolution

Upon reconnection, the state of both the client and the server are updated to reflect changes to shared data. In a multi-user environment, problems arise with such updates when users perform conflicting operations during disconnection. The simplest control policy is *pessimistic control* [3] where a single client is granted exclusive access to a data object during a disconnection. This is impractical since unreliable access could force all

other users to wait for an indefinite time period until the owner reconnects and relinquishes control. More common is an *optimistic control* [3] policy where access is granted to all users. This improves availability of shared data but allows the possibility of conflicting updates. It then becomes important to provide mechanisms to detect and resolve conflicts that arise during disconnected operation. Two mechanisms that attempt to solve such conflicts are operation based update propagation and isolation only transactions.

2.3.1. Operation Based Update Propagation

Partial connections place limits on the frequency of client updates sent to the server for large data objects. It is possible to express the changes to an object by recording a sequence of operations performed by the client on a data object. The client can send this sequence to a well-connected *surrogate client* [5] that performs the same operations on its own copy to generate the version held by the client. If the client and surrogate copies are identical then the surrogate commits the file over its high-speed link. Otherwise, the client itself transfers the file to the server on a slower connection. The advantage of such a system is the compact method with which changes to an object may be expressed, thereby saving network bandwidth and other resources. Additionally, use of a surrogate removes the need for the server to run long and possibly malicious computations. [5]

2.3.2. Isolation-Only Transactions

The *isolation only transaction* (IOT) [7] system extends CODA and Unix to provide detection and resolution of read/write conflicts arising from disconnected operation. Such conflicts are detected by recording modifications to objects during disconnected operation as a sequence of transactions. Upon reconnection, these transactions are compared to the server state using *global certifiability criteria*. If one views modifications to a file as a sequence of transactions they can be compared to all server transactions to that file to detect conflicts. [7] This is an improvement over CODA, since it allows the detection of both read/write conflicts in addition to write/write conflicts. If successfully validated, the transactions are committed to the server. Otherwise IOT will attempt to resolve the conflicts automatically. Recorded changes are resolved incrementally in serialized order to rebuild a correct result. The advantage of such a system is the transparent and automatic detection and resolution of conflicts. The problem is that the conflict resolution mechanism is only so robust and the system will default to manual repair in some situations.

3. The Hybrid Approach

While the analyzed systems and solutions all target their specific areas of focus well, one can combine them into a synergetic, hybrid combination to provide the advantages of each solution. This section describes how the systems can be used together as an elegant solution to mobile data access.

3.1. Disconnected Availability

Hoarding is most appropriate for preparing for a known disconnection, as it lets users explicitly choose what they want for the disconnection period. This shifts the burden of "blame" from the system to the users. Whatever files users cannot access during the disconnection period is their "fault". However, hoarding is inadequate for an unknown disconnection or network partitioning as nothing can predict a network outage or broken link. In that event, predictive caching is useful, since it continually monitors and analyzes the user's past computing history to automatically pre-fetch and cache files the user may want in the future.

A hybrid approach crossing the two paradigms is in order. One can combine the two by having the client maintain a hoard database of files the system and user explicitly want or need. The hoard can be explicitly checked when the user wishes to disconnect, and the system can then have a quick last chance to get any files it needs off the server before the disconnection occurs. A *hoard walker* [3] approach, as in the Coda, can still be used to occasionally restore equilibrium. All this can be used in conjunction with the system's pre-caching policy. The client can maintain usage histories of applications over a time period, and help the hoard walker to maintain a balanced cache that meets both the current and future computing needs. The hoarding and pre-caching systems complement each other, but are not intimately tied together. This allows them to be tuned separately to achieve a fine balance between preparedness and performance.

Another approach is to combine the two directly into one system whereby the user, instead of specifying files in the hoard database, can instead specify their activities or projects. Instead of saying, "I want file1.txt, file2.txt, etc.", users can tell the system that they want to be able to work on the current fiscal year's budget, or they want to be able to look at a photo album of their last vacation. This approach is a more high-level approach, and allows the user to think around "activities" rather than low-level individual files. The system can use the access tree approach to associate computing histories with applications or projects. This way, the user specifies explicitly what applications or projects they might need in the period of disconnection, and the system will predictively cache the files needed for that project (as determined by the access tree of that project).

3.2. Partial Connectivity Adaptation

A hybrid approach is effective for joining the Prayer File System consistency approach and the Odyssey adaptation system. As with the disconnected approaches, PFS and Odyssey complement each other and work together to gracefully handle a partially connected environment.

For instance, using the hybrid approach, applications can allow a finer granularity for file requests or file propagation. For example, an application starts out on a fully connected network. The application can request copies of entire files and transfer whole files back and forth between the server. If the client is taken off the fully connected network and moved to a wireless connection, then the application only has a partial connection to the network. The operating system should then notify the application that the network has changed, thus allowing the application to reduce itself to finer granularity

accesses (as per the Prayer File System). Applications can request to get copies of only certain parts of a file, leaving the rest of the file on the server. They can then modify their parts and then send back update parts slowly in the background using a background daemon for the server to replay. This makes for efficient use of the available network since only essential data is sent back-and-forth.

Additionally, allowing applications to adapt to network environment changes via notifications gives them the power they need to achieve their best performance. For example, when network bandwidth changes, clients can respond by altering the resources they require or algorithms they use in order to perform well under the new environment. With the use of finer granularity access, applications can choose to request varying amounts of data, depending on the network connection and the performance they desire. Some applications may choose more data; hence more download time for better correctness. While for other applications, real time delivery is more important than all the data and will choose download less. However, the essential point is that the combination of both methods gives the power to the application to decide how to adapt. After all, only the application can determine how it is best served under the new environment.

3.3. Update Propagation and Conflict Resolution

A combination of the presented approaches of update propagation and conflict resolution provides the best performance, robustness, and transparency.

When the client performs changes to a file, those changes are recorded as a sequence of operations and sent to the server when a connection becomes available. The server will then attempt to verify the changes and resolve any conflicts. If the server successfully finds a conflict free sequence it will build a new file from it to replace the old copy. Otherwise, the user will be alerted to manually correct the problem.

Both the operation-based update propagation and isolation only transactions have their limitations. A limitation of operation-based update propagation is that it is only capable of detecting if the local copy and the global re-generate copy were the same. This suffices for partial connections where updates can be sent immediately to the server. However, during disconnected operation, the threat of more complex conflicts grows, which requires a more sophisticated mechanism. Using isolation-only-transactions, servers can detect subtle conflicts such as write/write and read/write conflicts.

Because both approaches use sequences of operations as the representation for changes to a file, and important concern is where to perform the re-building of a file. Operation based update propagation uses a surrogate client. The surrogate client has access to a high-speed link to the server. This shields the server from additional computation and security threats. However, it adds another link, which may become disconnected. This would require some method of finding a connected surrogate or requiring the surrogate to have some disconnected access mechanism of its own. The isolation-only-transactions system requires that the client re-build the file. This is not always feasible since a server

copy would need to be sent over a possibly slow connection thus defeating any attempt to limit bandwidth consumption.

Since neither the client nor surrogate is a suitable site for conflict detection, it would seem that the server is the best location to re-build the file. Many mobile devices lack computational power and may attempt to offload computation to the server. It is reasonable then to ask the server to handle the propagation of updates. Furthermore, questions of security must be addressed in any case when offloading to a server so updates are not a unique security issue.

4.0. Conclusion

The presented approaches to mobile data access can be used collectively to provide a more efficient and better performing environment to the mobile client. The systems presented in this paper complement each other to produce a new, cleaner, elegant and more responsive hybrid system that adapts itself to changing network environments.

This hybrid system permits the mobile client to be better prepared for disconnected operation, whether it is an expected or unexpected disconnection. The users may explicitly specify the files they need in times of disconnection. The client will also monitor access history and usage patterns in an attempt to predict the user's working set, taking into account their attention shift history. Furthermore, this allows the hybrid system to predictively cache files that the users may want in times of disconnection. The users can also specify generalized tasks they may want to run while disconnected, from which the client can analyze previously archived "runs" of those tasks to ensure that the needed files are cached on the client.

In addition, this hybrid system also provides applications notifications of changes in the network environment and quality of the connection. Applications can adapt to the new network environment by adjusting their required resources and changing their algorithms for the best performance. After all, only the application can determine what data is best for their performance. Moreover, by using fine grain access at the record level, applications can further specify what essential data is downloaded, thus decreasing the performance impact on the network environment.

Lastly, such a hybrid system allows updates can be propagated back to the server efficiently. By bridging the inefficiencies of both the isolation-only transaction system (building the updates on the client), and operation-based update propagation system (requiring the connectivity of a surrogate server), the system can be more tolerant of updates occurring during a disconnected period.

References

[1] Dwyer D. and Bharghavan V..
A Mobility-aware File System for Partially Connected Operation.
ACM Operating Systems Review, 31(1):24--30, Jan. 1997.

[2] Huston L.B. and P. Honeyman.
Partially Connected Operation.
Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing, pages 91--97, 1995.

[3] Kistler J. and Satyanarayanan M..
Disconnected Operation in the Coda File System.
ACM Transactions on Computer Systems, pages 6(1):1-- 25, February 1992

[4] Kuenning G. H., Popek G.J., Reiher P.L.
An Analysis of Trace Data for Predictive File Caching in Mobile Computing
In 1994 Usenix Summer Conference, April 1994

[5] Lee Y., Leung K., and Satyanarayanan M..
Operation-based Update Propagation in a Mobile File System.
Proceedings of the USENIX Annual Technical Conference, June 1999.

[6] Lei H. and Duchamp D.
An Analytical Approach to File Prefetching.
In 1997 USENIX Annual Technical Conference, January 1997

[7] Lu Q. and Satyanaraynan, M.
Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions.
Proceedings of the fifth workshop on Hot Topics in Operating Systems, Orcas Island, Washington, May 1995

[8] Noble B.D., Price M. and Satyanarayanan, M.
A Programming Interface for Application-Aware Adaptation in Mobile Computing.
Proceedings of the Second USENIX Symposium on Mobile and Location Independent Computing, Feb. 1995

[9] Noble B. D., Satyanarayanan M., et al.
Agile Application-Aware Adaptation for Mobility.
In Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (St. Malo, France, October 1997).