

Pre-Processing Directory Structure For Improved RSYNC Transfer Performance

Alireza Ghobadi*, Ehsan Haji Mahdizadeh*, Yong Lee Kee**, Li Kok Wei**, Mohamad Hossein Ghods***

**Faculty of Information Technology Multimedia University Selangor, Malaysia*

alireza.ghobadi@mmu.edu.my, ehsan.mahdizadeh@mmu.edu.my

***Intel Microelectronics (M) Sdn. Bhd., Penang Design Center, Penang, Malaysia*

leekee.yong@gmail.com, kok.wei.li@intel.com

****SOHA Sdn. Bhd., ABI- Malaysia Technology Commercialization Centre, Malaysia*

ghods@soha.com.my

Abstract—This study deals with the problem of maintaining large hierarchy folder replicated in a distributed environment. It was found that this problem afflicted a number of important applications, such as synchronization of Hierarchy Folders between Peer-to-peer environments, synchronization of data between accounts or devices, content distribution and web caching networks, web site mirroring, storage networks, and large scale web search and mining. At the core of the problem lay the File in Hierarchy Folder synchronization challenge. This challenge posed the question; “Given two versions of files inside Folders on different machines, call outdated hierarchy and a current one, how can we update the outdated version with a minimum communication cost, by exploiting the significant similarity between the versions?” Although a popular open source tool for this problem called RSYNC is being used in hundreds of thousands of servers around the world, only very few attempts have been made to improve this tool in practice.

In this paper, we propose a framework for remote file synchronization and describe several new techniques that result in significant time savings and less error during files transfer. Our focus is on applications that involve very large collections of Hierarchy Folders including files and folders. We show that a prototype implementation of our framework and techniques achieves significant improvements over RSYNC. As an example application, we focused on the capable synchronization of large folder transfer between peer-to-peer distributions. This method is provided for synchronizing folders between the server which is updated by some client and other servers. Our algorithm adopts the Sender/Receiver Server structure based on a client modification.

Keywords – RSYNC, File Transfers, Synchronization

I. INTRODUCTION

Many problems are faced by semiconductor companies with globalized R&D offices where hundreds of research, development, production and test engineers across the globe are engaged in. This type of environment requires good efficient management of data and especially the need to conserve network bandwidth usage to drive cost down whilst ensuring productive work done. Further complications were seen in a shared network where other corporate communications, data transfer and even voice traffic that required some quality of service. Hence, the ever growing data bandwidth is limited. The study deals with types of data that needs to be shared are schematics and other design data that

runs into terabytes in the situation wherein the synchronization of 2 servers across very distant locations and the length it takes to accomplish that synchronization.

Some huge companies such as Yahoo, has a distributed service and several servers on the world as well. Data and information on these servers need to be shared or synchronized. A Distributed File System (DFS) is a network file system where data can be distributed across several physical computer nodes [3]. File system objects are named in a hierarchical name space that is distributed among multiple servers [3]. A distributed file system enables programs to store and access remote files exactly as they do on their locations, allowing users to access files from any computer on the intranet from a local area network (LAN) or even on a wide area network (WAN). All DFS systems pass through procedures of Authentication, Updating, versioning, Synchronization, and Replication of data.

There are many tools to solve synchronizing data between different machines. Some of them, such as Microsoft's ActiveSync, or Palm's HotSync, are used to synchronize data between a desktop or online and a mobile device. They typically transfer the entire file if a change has occurred. Though for record-based data such as appointments and contacts only updated records are transmitted in most cases. But this kind of synchronizations is not suitable if it involves the transfer of huge data and distance between two devices, or insufficient bandwidth. Even with sufficient bandwidth it the process would take a long time and in disconnection crisis, it would be necessary to replicate all of the files again.

There are other tools designed based on partial synchronizations of the files. Partial synchronizations check any updates and then just transfer part of the file or folder which has been changed. RSYNC is a famous tool to synchronize data in file synchronization of Folders Synchronization [3][4]. The proposed *RSYNC* file and Hierarchy Folder synchronization tool is currently commonly used between different machines, to mirror web and ftp sites, and to perform backup over a network.

There are a number of applications designed for the above scenario to solve synchronization files and deep HF between two machines, remote backups, mirroring of large ftp, content distribution, and web search engines. In this case, instead of sending the entire updated files, it would be desirable to be able

to perform the updating by sending only a limited amount of data compared to the change between the two versions.

In this paper, focus was on updating the deep HF updating problem; referred to as HF synchronization problem. This work is primarily motivated by several applications in large scale distributions, but the techniques are applicable in more general cases. The author believes that file and HF synchronization is a fundamental operation in distributed systems. RSYNC is a famous open source software tool which is connected to this problem. There are some papers which described RSYNC [1][2]. The goal of this study is to derive techniques that achieve significant savings over RSYNC particularly in the case of large collections. In this paper, an algorithm for solving replication, and a substantially improved updating HF on peer to peer system in Distribute environments was developed. The proposed solution to this study's problem was to update hierarchy folders between two Servers.

II. RELATED WORK

The RSYNC protocol daemon uses the default TCP port 873 for transferring files. RSYNC supports local directory synchronization, handshake process, data synchronization and error detection [5][6]. At the startup process, the sender entity transmits file details to the receiver entity [7]. During the file transmission process, the receiver entity will act as *generator* and create the block checksums for the file and pass the block checksums to the sender entity. Upon receiving the block checksums from the receiver, the sender entity will read the local file and generate its own block checksums. The two sets of block checksums will be compared, one block at a time (*rolling checksum*), and all the resulting matching and non-matching data will be sent to the receiver entity together with the whole file-checksum. The receiver entity rebuilds the file based on this matching information and generates a new file-checksum which will be compared with the sender's file-checksum for ensuring the integrity and reliability of the file [7][8]. Therefore, the overall file transfer time via RSYNC is influenced by the computation time for checksum, hash table, transmission time for checksum, hash table and the actual file.

According to Figure 1, RSYNC synchronizes remote directory hierarchies between two servers (Sender and Receiver servers). The hierarchy folder exists on the sender receiver. In this example, the client updates some files or folders existing on the receiver server's hierarchy. Then, the Sender Server checks and transfers them to receiver server. The important part in this scenario was found to be in the modification or updating mechanism which RSYNC uses. There are two problems that arise during the RSYNC process. First, RSYNC tried to check folders and files one by one which took a long time and the checksum which is normally not the same on the sender and receiver sides.

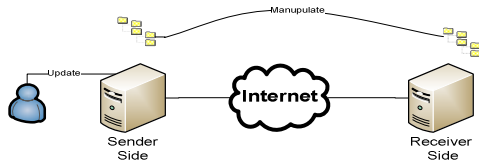


Figure 1: Environmental setup for testing Hierarchy.

The basic approach in RSYNC is to split a file into blocks and use hash functions. These hashes are sent to the other machine. One issue is the lack of alignment between matching blocks in two files. In the receiving side, when comparisons are made on the received hashes with the other file, it was found that all substrings were of the same size. For efficiency, hashes are composed from two different hash functions, a fast but unreliable one, and a very reliable one that is more expensive to compute.

There are several researches that tried to solve RSYNC hashing and delta compression problem by changes of the techniques [16]. Other researches tried to find solution by wrapping RSYNC and reduce complexity of hashing table which is good for a file transferring on Peer-to-Peer environment but still these is weakness on the Hierarchy folder synchronization on RSYNC [17]. All of these researches about reduce time of single file transferring or multiple file transferring.

Another problem of RSYNC, is called Delta compression. Delta compression is the solution to encoding one file relative to another similar file/folder, where both files/folders are available during the encoding. Thus, file synchronization can be used when the two files are located at different machines. But this compression has significantly higher cost. Some research on the open source described delta compression [12][13][14], and gave an overview of delta compression and file synchronization techniques. There are some applications designed by these techniques [11].

The RSYNC file synchronization algorithm was proposed by Tridgell and MacKerras [4][5]. This algorithm, looks like many of the theoretical algorithms and assumes that a hash function is reversed as part of the decoding operation; while this is acceptable communication complexity [6]. An exception are algorithms proposed in [3][5][7][8] that are based on recursive partitioning of blocks similar to our implementation. These algorithms can also demonstrate some complex problems. The literature review also revealed some limited experimental results [9][10]. This algorithm has space requirement and complexity problems. Recent work in [11] presents a version of the RSYNC algorithm that updates files in-place without using additional temporary space.

Hash-based techniques used in RSYNC was proposed by Karp and Rabin [15] to partition a data stream into blocks in a consistent manner on both sides of a communication link, and then send hash values to encode repeated substrings.

A characteristic feature of RSYNC (and thus RSYNC+) was the use of block-level check summing to perform differential file update when an existing file has been updated, or replicated i.e., has a new disk image under the same file name [18].

Now, based on the abovementioned items, a definition of the file synchronization problem and the reduction of some over heads by a hash table is supplied. Figure 2 show the HF information structure.

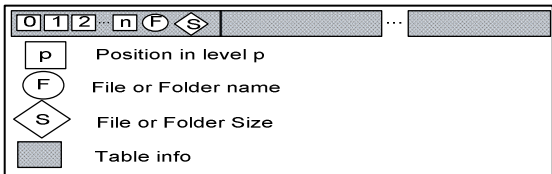


Figure 2: HF information structure.

$$C[i] = N[0] - N[1] - \dots - N[n].F * \text{Size} \quad (1)$$

III. ALGORITHM AND IMPLEMENTATION

The proposed Hierarchical Folder Synchronization Algorithm (HFSA) works as a middle ware that reduces the computational time required by RSYNC at both the sender and the receiver nodes. It is a pre-processing procedure that reduces the number of folders and files for RSYNC to assess. Without the pre-processing, a standard RSYNC will traverse the whole folder structure and when there are changes to only a very limited number of folders, it will be an unnecessary overhead. This will also mean that on the receiver node, less checksum computation is required by RSYNC.

The HFSA consists of two parts; (1) the HFSA at the sender node that focuses on creating the hash table and transmitting this file and (2) the HFSA at the receiving node that focuses on receiving the hash table receive the information on the changes and re-construct the respective folders.

A. HFSA at the Sender Node

Figure 3 presents the architecture of the HFS at the sender entity.

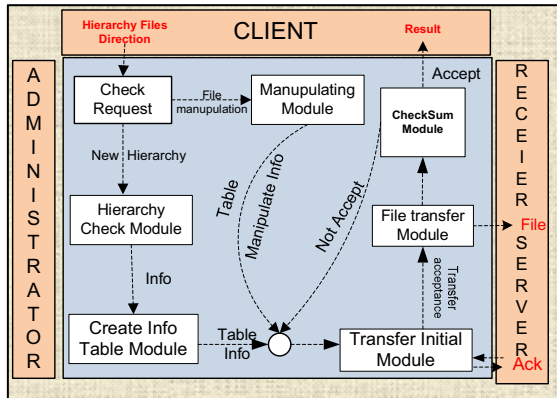


Figure 3: HFS Algorithm architecture at the sender entity.

A request needs to be sent to the HFSA module to initiate it. A check on the changes to the file or folder is conducted and if changes are detected, this module will send a message consisting of the file/folder names and the location of the file/folder to the *Manipulating Module*.

If the whole hierarchy synchronization is requested, then a hierarchy check by *Hierarchy Check Module* is done. This module is responsible to check that the Hierarchy folder exists or not. If it exists then it will create a Tree Hierarchy Folder (THF).

Data that is created in this module is inclusive of files and folders and the location in the THF. This data is then sent to the *Creating Table Module*.

We assume that the level of the hierarchy structure is to address many levels and the naming convention is to use ‘-’ to indicate each level. Names of files and folders are attached to the end of the levels indicator separated by ‘.’ and finally we use the symbol ‘*’ to separate the file size. If the file is a folder, the size is defined by “-1”. Note that we are aware that the separator symbols may conflict with file and folder names. However, for this research, we will assume that file and folder names will not include ‘*’. The naming can be represented by;

Where n is the hierarchy level, F is the file or folder name and Size file size (or -1 if it is a folder).

This information structure is stored in a one dimensional array where the benefits are obvious. Firstly, the single dimension array is sufficient to store the whole hierarchy and secondly, this array is sufficiently small and hence will not require much network bandwidth to be transmitted over the network.

The next module is the *Transfer Initial Module*. This module is responsible for transferring the “*Data Tree Information Table*”. In doing so, it needs also to include a function to monitor the transfer. This module will expect an acknowledgment from the receiver side within a stipulated time (t_1). If the acknowledgment is not received in t_1 , the *Transfer Initial Module* is re-activated to check the last file transfer and resend it. It basically works as a timeout. This time t_1 is computed using historical knowledge of the actual transfer time (t_2) and also the time it requires to read the HFS time (t_1) with some buffer ($0.1 t_1$). This timeout is represented by;

$$t_i = t_2 + t_1 + (t_1 * 0.1) \quad (2)$$

Once the *Transfer Initial Module* receives the acknowledgment within the stipulated timeframe and the actual file transfer is done by the *File Transfer Module*. A sample implementation is shown in figure 4 that illustrates a hierarchy folder structure with 7 Levels. Each level includes a particular number of folders and files. In the sample, the total number of files and folders to be involved in this sample is 27. Included in the sample are files with the same names located in different folders

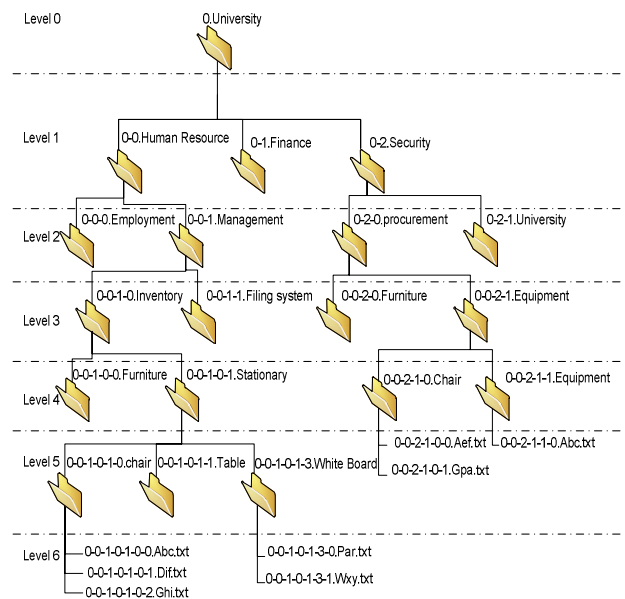


Figure 4: Examine Hierarchy Folder.

Using our sample in figure 4, we will try to illustrate the HFSA. The HFSA will initially discover the number of levels in the hierarchy, the number of files and folders in this hierarchy and create the numbering for the various levels. The

THF is then created by *Hierarchy Check Module*, as per table 1. For example, the Employment Folder is in the folder “University/Human Resource/” and this is indicated by “0-0-0.Employment*(-1)”. This creates a unique number which will avoid duplication problem.

LEVEL	Array Number	File or Folder
0	N[0]	0.University*(-1)
	N[1]	0-0.Human Resource*(-1)
1	N[2]	0-1.Finance*(-1)
	N[3]	0-2.Security*(-1)
	N[4]	0-0-0.Employment*(-1)
2	N[5]	0-0-1.Management*(-1)
	N[6]	0-2-0.procurement*(-1)
	N[7]	0-2-1.University*(-1)
	N[8]	0-0-1-0.Inventory*(-1)
3	N[9]	0-0-1-1.Filing system*(-1)
	N[10]	0-0-2-0.Furniture*(-1)
	N[11]	0-0-2-1.Equipment*(-1)
4	N[12]	0-0-1-0-0.Furniture*(-1)
	N[13]	0-0-1-0-1.Stationary*(-1)
	N[14]	0-0-2-1-0.Chair*(-1)
	N[15]	0-0-2-1-1.Equipment*(-1)
	N[16]	0-0-1-0-1-0.chair*(-1)
	N[17]	0-0-1-0-1-1.Table*(-1)
5	N[18]	0-0-1-0-1-3.White Board*(-1)
	N[19]	0-0-2-1-0-0.Aef.txt*(254)
	N[20]	0-0-2-1-0-1.Gpa.txt*(340)
	N[21]	0-0-2-1-1-0.Abc.txt(540)
	N[22]	0-0-1-0-1-0-0.Abc.txt*(623)
6	N[23]	0-0-1-0-1-0-1.Dif.txt*(231)
	N[24]	0-0-1-0-1-0-2.Ghi.txt*(1225)
	N[25]	0-0-1-0-1-3-0.Par.txt*(756)
	N[26]	0-0-1-0-1-3-1.Wxy.txt*(987)

Table 1.THF Table Information.

After the table is created, we add the “Request Tag” (TG) to the header of the THF. This tag is necessary for the communication between the sender node and the receiver node as it is used for the acknowledgment in the *Transfer Initial Module* and other communications. The tag is as depicted in figure 5.

TG	N[0]	N[1]	N[2]	----	N[26]
----	------	------	------	------	-------

Figure 5: Message which created by Hierarchy check Module.

Once the acknowledgment is received, the *File Transfer Module* will use RSYNC to achieve the transfer. After each transfer, the *Checksum Module* conducts the check on the integrity of the file transfer and if it is found to be compromised, then it will request for the transfer to be re-initiated. After the last file is sent, the system will send a summary to the client.

B. HFSA at the Receiver Node

The architecture of the HFSA at the receiver entity is presented in Figure 6. On the receiver node, the HFSA is initiated at the *Request Checking Module*, where it receives the request message from the sender node. If the request contains a message to create a new hierarchy folder, the *Folder Creation*

Module prepares the folders to store the file to be received. This creation folder is related to the *THF tables*, as shown in figure 6. The folders will be created as per the message received. An acknowledgement message with the Request Tag (TG) will be sent to the sender node, indicating successful receipt of the message and to request the sender node to start the file transfer process.

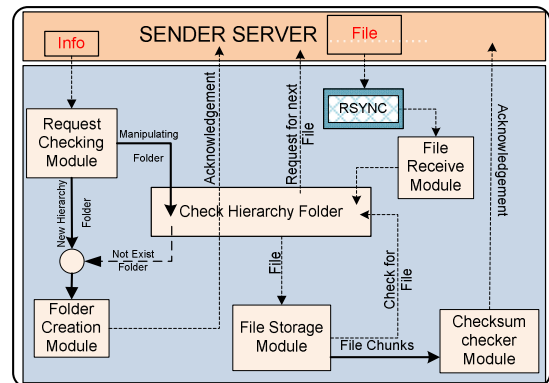


Figure 6. HFSA Algorithm architecture at the receiver entity.

The *File Receive Module* receives the sender’s file via

TG	N.F[0]*Z[0]	N.F[1]*Z[1]		N.F[n-1]*Z[n-1]
----	-------------	-------------	------	-----------------

Figure 7: Received Message with the Request Tag (RT)

The RSYNC receiver node will then be initiated to wait for the RSYNC transfer from the sender node through the *File Receive Module*. The *File Storage Module* will then refer to the *file storage table* and store the file in appropriate folders according to the predefined THF table that was created earlier. When all the files are received and stored appropriately in a particular file, the HFSA will pass the file folder to the *Checksum Checker Module* for calculate check sum and send this to the sender node.

IV. RESULTS AND ANALYSIS

The HFSA was implemented using the Java programming language with a mix of UNIX system calls, RSYNC for transferring of files and Java socket programming for both the server node and the receiver node.

We did a preliminary test using 3 files of various sizes up to 2 GB in a simulated environment with 100 levels of folder hierarchy that contains 10,000 files. In our test, we compared HFSA (with RSYNC) and standard RSYNC with modifications to the files at different levels in the hierarchy. An initial test was conducted; to compare the total number of files being updated when using HFSA and standard RSYNC.

A. Total Number of Files Transferred

In synchronization, the number of files being updated by RSYNC is dependent on the actual number of files that has been modified and needs synchronization, information about the folders and also the location of the changes made within the files.

We modified 520, 1040, 2080, 4160 and 8320 files (out of the possible 10,000 files) and captured the number of files

being transferred from the output of RSYNC. Figure 8 shows the resulting synchronization rates where the rate (R) is calculated based on the number of the updates on the Hierarchy Folders (N_H) divided by number of the files updated by the tool (N_T).

$$R = N_H / N_T \quad (3)$$

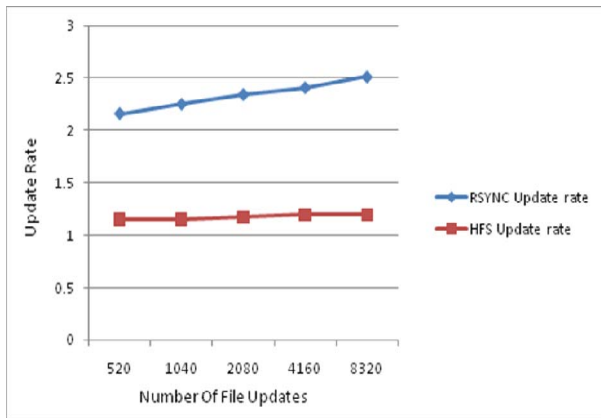


Figure 8: Comparison Table Base on Updating Rate.

From our preliminary observation (Figure 8), the number of files being updated by the standard RSYNC is almost twice that when HFSA pre-processing was used. Another observation is that with HFSA, the ratio of actual files being transferred against the number of files modified seems to remain constant no matter how many files were modified. This is in comparison with the standard RSYNC where this ratio also increases with the number of files that requires synchronization.

Further study may be required to ascertain that the HFSA reduces the checksum activity when comparing files and folders. This is because, the HFSA have already transferred the THF and hence less computation required at the receiver node.

With less number of actual files being transferred, the overall time of synchronization is also expected to be improved.

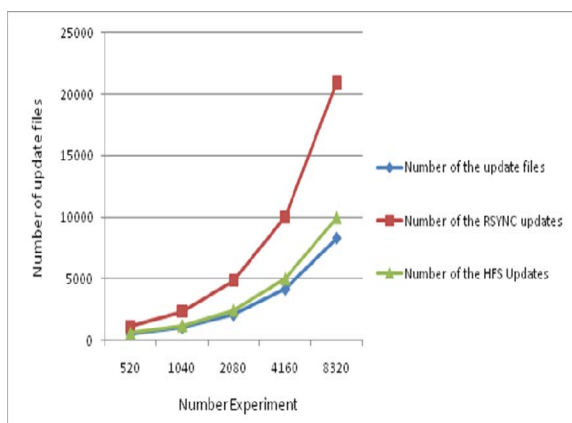


Figure 9: Comparison Table Base on Number of Updated Files.

Figure 9 depicts a graph of the number of updated files in RSYNC and HFSA and compares it to the number of files that needs to be updated. This is a non-normalized view. Figure 10, shows that HFSA can reduce the number of actual files being updated by up to 48%.

V. CONCLUSION AND FUTURE WORK

We have not done conclusive work on the HFSA but it is a promising method to improve overall performance when using RSYNC. There are some limitations with our initial tests in that (1) the files being modified is not a real sample of any real world situation and more importantly (2), we have not done any timing measurement on the overall time needed for synchronization. For the second part, synchronization time for the standard RSYNC should include checksum calculation of the transferred hash table, comparison of the checksum transfer, and the file transfer time. In the synchronization time for HFSA, we have also included the overheads for the hierarchy folder calculation, time of THF transfer in sender and receiver nodes, and file transfer time.

It should also be noted that with a smaller number of hierarchy levels (instead of the 100 we use in our initial test), the overall effect may not be significant.

This implementation of HFSA does not require any changes to the RSYNC protocol and hence can be easily used whilst retaining all of RSYNC features.

REFERENCES

- [1] A. Tridgell and P. Mackerras, "The rsync algorithm," <http://rsync.samba.org/>, Nov 1998.
- [2] A. Tridgell, "Efficient Algorithms for Sorting and Synchronization", PhD thesis, Australian National University, 1999.
- [3] J. Fridrich and M. Goljan, "Robust hash functions for digital watermarking," International Conference on Information Technology: Coding and Computing, pp. 178–183, 2000.
- [4] B. Coskun and B. Sankur, "Robust video hash extraction," IEEE 12th Proceedings on Signal Processing and Communications Applications, pp. 292–295, 2004.
- [5] Y. Lin, D. Varodayan, and B. Girod, "Image Authentication and Tampering Localization using Distributed Source Coding," IEEE 9th Workshop on Multimedia Signal Processing, pp. 393–396, 2007.
- [6] S. Draper, A. Khisti, E. Martinian, A. Vetro, and J. Yedidia, "Using Distributed Source Coding to Secure Fingerprint Biometrics," IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 2, 2007.
- [7] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-checkcodes," Information Theory, IEEE Transactions on, vol. 47, no. 2, pp. 619–637, 2001.
- [8] C. Yeo, P. Ahammad, and K. Ramchandran, "Rate-efficient Visual Correspondences using Random Projections," to appear in IEEE International Conference on Image Processing, 2008.
- [9] R. Puri, A. Majumdar, and K. Ramchandran, "PRISM: A Video Coding Paradigm With Motion Estimation at the Decoder," IEEE Transactions on Image Processing, vol. 16, no. 10, pp. 2436–2448, 2007.
- [10] B. Girod, A. Aaron, S. Rane, and D. Rebollo-Monedero, "Distributed video coding," Proceedings of the IEEE, vol. 93, no. 1, pp. 71–83, 2005.
- [11] J. Langford. Multiround rsync. January 2001. Unpublished manuscript.
- [12] J. MacDonald. File system support for delta compression. MS Thesis, UC Berkeley, May 2000.

- [13] A. Tridgell. Efficient Algorithms for Sorting and Synchronization. PhD thesis, Australian National University, April 2000
- [14] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 117–128, May 2000.
- [15] N. Ramsey and E. Csirmaz. An algebraic approach to file synchronization. In Proc. of the 9th ACM Int. Symp. on Foundations of Software Engineering, pages 175–185, 2001.
- [16] D. M. Lopez-Pacheco, “Enabling large data transfers on dynamic, very high-speed network infrastructures”, 2007.
- [17] C. Barakat, E. Altman, W. Dabbous. “On TCP performance in a heterogeneous network: a survey”. IEEE Communication Magazine, January 2000.
- [18] T. Suel, P. Noel, and D. Trendafilov, “Improved File Synchronization Techniques for Maintaining Large Replicated Collections Over Slow Networks”, ICDE 2004, March 2004.