The Design of a Storage Architecture for Mobile Heterogeneous Devices

Glenford Mapp
Networking Research Group
School of Computing Science
Middlesex University
Email: g.mapp@mdx.ac.uk

Dhawal Thakker
Networking Research Group
School of Computing Science
Middlesex University
Email: d.thakker@mdx.ac.uk

David Silcott
Networking Research Group
School of Computing Science
Middlesex University
Email: d.silcott@mdx.ac.uk

***Abstract*—Mobile computing devices such as smart PDAs and ultra-light laptops with several networking interfaces are becoming commonplace. The provision of networked data storage facilities will greatly extend their use. This paper looks at the design of a storage architecture for such devices. A two-level structure is proposed in which one component, the mobile memory cache (MMC), moves when the node is mobile. A prototype MMC was designed and evaluated. Preliminary results are presented which show that the system should be able to provide a high-performance service.**

I. Introduction

Mobile computing devices such as smart PDAs and ultra-light laptops are becoming commonplace. In addition, these devices now have several networking interfaces including high-speed LAN, WLAN (802.11a/g) and 3G networks. Users of these devices will expect to be always connected, with seamless switching between available systems. This is being made possible by the development of an architectural framework for heterogeneous networking with support for vertical handovers [1]. Global, ubiquitous access will allow new services to be delivered to these devices, enhancing the lives of their users.

One area of interest is the provision of network data storage to mobile devices. There are several reasons for this: the first is security. Data held on mobile devices is inherently insecure as these devices can be easily lost or stolen. System administrators are therefore very reluctant to allow data of any significance to be held on these devices especially if the use of these devices is not restricted to specific geographical locations. With access to data storage using a secure link, nothing is compromised if the end device is lost or stolen since the device need not contain persistent storage. The second major reason is ecological. Laptops can easily carry hard disks; however, this tends to significantly increase their power requirements, noise profile and price. In addition, good quality storage using solid state disks is still prohibitively expensive. Finally, access to data networks must be available to the global population to enhance the economic development of poorer countries. Presently, laptops and other mobile computing devices are simply too expensive for the masses in the Third World. Access to global storage using fast interfaces will allow cheaper and more reliable devices to be built.

This research direction is also justified by the development and deployment of high-speed networks. Network interfaces of 1Gbps are now commonplace and are fairly inexpensive. Work is now proceeding to bring 10Gbps to the desktop. Secondly, we are seeing the deployment of faster wireless network interfaces. For example, most devices now have 802.11g interfaces which have over-the-air (OTA) rates of 54 Mbps with Media Access Control Layer, Service Access Point (MAC SAP) rates of 25 Mbps. However, the next generation wireless technology, 802.11n, is being developed. This promises to deliver over 540 Mbps OTA speeds and MAC SAP speeds of 200Mbps [2]. It should be noted that at these speeds, it is commonly faster to get data from the memory of a remote machine than from a local hard disk [3]. Finally, we are finally seeing large investments in broadband access networks which will allow over a 100 Mbps to residential networks. This has facilitated a greater demand for flexible working patterns, making support for mobility a key requirement of future systems and services.

Substantial research in distributed storage systems concentrated on the use of storage servers connected via a fast local area network. This has given way to commercial systems such as NFS or iSCSI [4]. We have also seen the development of Storage Area Networks (SANs). It should be pointed out that these systems whilst useful in a local area network do not usually scale well over wide-area networks, resulting in poor performance in these environments. Hence a key requirement is therefore to allow mobility while trying to maintain good performance.

A new development in this area is the ability to support the migration of lightweight services. This has been made possible by the specification of the Context Transfer Protocol (CXTP) detailed in RFC 4067 [5]. This new facility has been backed up by the increasing use of machines acting as network management servers. These servers tend to have a significant amount of memory and or disk storage. This deployment is now taking place in the home as home networking becomes commonplace.

This paper looks at the design of a storage architecture for mobile heterogeneous devices and tries to address some of the challenges detailed above. What is being proposed is a two-level service architecture in which one component is a **mobile memory cache (MMC)**. The MMC makes use of **persistent**

storage servers (PSS) to provide persistent storage. We detail the implementation of a MMC running over a 1Gbps network and preliminary performance results are given. The rest of the paper is outlined as follows: Section II examines the design choices for the storage infrastructure while Section III details the key design structures required to implement the system. Section IV highlights the current work being done including the Middlesex testbed. Section V displays preliminary results and Section VI discusses related work. The paper concludes in Section VII looking at future work.

II. The Design Choices for the Storage Infrastructure

There were some key decisions in the design of a storage architecture. These are discussed below:

File vs Block Storage: Since we are interested in a portable system, it was felt that the actual service provided by the server should be as simple as possible. This meant that system should be storing blocks of data and not files. It was also felt that managing blocks gave the system the best flexibility in terms of client access as no access abstraction, such as a file abstraction, is imposed which must be supported at the server end. The storage system will have no idea how data in the blocks is being used or accessed; this is the responsibility of the client. So if the client was a file system, the actual file system would run on the client and not the server. This decision also meant that blocks would have to be independently identified and that there must be a way of maintaining and guaranteeing the identity of a block.

Support for Mobility: A key requirement is that the users should not experience a significant loss in performance when they are mobile. This will require that users are always connected to the storage facilities and that there is a way to maintain performance of the system. The authors believe that this is possible by dividing the architecture into two components. The first is the use of an MMC which runs on a network server in the same network as the heterogeneous device. When the device enters a new network then the MMC is migrated to the local network so as to maintain the performance of the network. The second component is the use of a Persistent Storage Server or PSS. These servers provide permanent backup for the Mobile Memory Cache. In fact an MMC can backup data to two or more PSS machines so providing system redundancy.

An Independent Interface: It was decided to use an independent interface rather than to structure access to the server by clients using a specific interface, such as a SCSI interface. The main reason for this is flexibility. Since the system will be used by applications as well as system utilities such as file systems, if there is some benefit in additional structure of access, this would be known by client of the system and a suitable wrapper could be written.

Support for encryption: In order to support a secure system, there needs to be support for transporting encrypted data. In a two-level system in which there is an MMC on the local network, a lower level of encryption may be required between the MMC and the mobile node if the LAN is considered secure. However, strong encryption is required to move data from the MMC to the Persistent server and vice-versa as this involves going over a wide-area network.

Support for Reliable Multicast Transport Services: As we want to back up blocks using more than one Persistent server at the same time, then we need support for reliable multicast mechanisms. So when the MMC needs to store a block it makes a multicast call which simultaneously contacts all the relevant servers.

Sharing: In order for clients to build distributed systems using this infrastructure, for example a distributed file system, there needs to be some support for sharing. We believe that support for this should be efficient but minimal at the block level. Hence at the block level, the system should support owner, read and write privileges. The owner of a block can read, write and delete the block. In order to read a block by someone who is not the owner, a read capability is required. Similarly, a write capability is required to write to the file by someone who is not the owner.

Coherency Algorithms: These are services on the client that help the client implement cache coherency. When a client wishes to share an object with a group of users, it asks for system to generate read and write capabilities for the object. A very simple scheme has been initially proposed in which readers or writers are given a specific time to read or to write to the object.

III. Design Structures

This section looks at key design structures required to implement the system described above.

A. Block Structure

Since it was decided to use blocks, the block structure is discussed in this section. A block of data on an MMC is represented as a unique 64-bit entity called the BlockID. The first 48-bit of the BlockID identifies the block while the next 16 bits is a security tag. The security tag is generated using several components including random bits and the time the object was created. All the blocks of an object have the same allocation information including the same security tag. The security tag is used by the MMC to check that the BlockID has not been changed. Read and write capabilities are generated for the object using another random function with the security tag of the object as the input as shown in Fig. 1.

Each client of the system has a unique ClientID which is used to identify itself to the service. The ClientID must be used in all calls to the MMC server to create, read from, write to and delete objects, which are represented by sets of BlockIDs. The client which creates an object is designated the owner of the object. Clients can also read and write to individual blocks of the object. Only the owner of an object can also ask the system to delete all or part of an object. In order to share objects with other users, owners can also ask for read and write capabilities to be created for an object. Similarly owners can revoke the read and write capabilities for an object it possesses. Finally, it

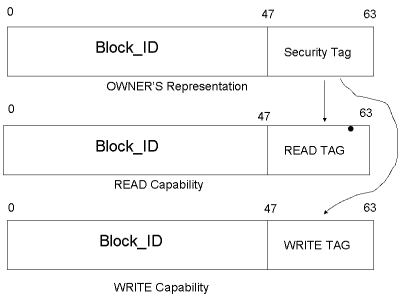

Fig. 1. Showing the BlockID Structure and Derived Read and Write Capabilities

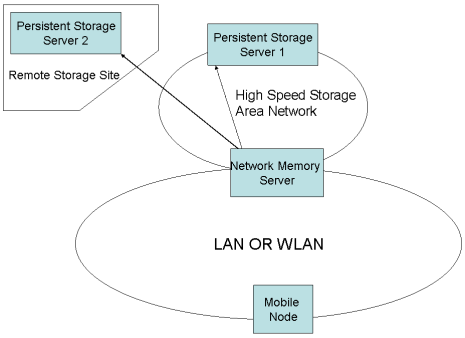

Fig. 2. Basic Work Scenario

is possible to pass to ownership of an object to another client of the system.

B. Support for Mobility

In this section, we show how mobility is supported. We first look at some common scenarios. Fig. 2 shows the basic environment at work. The mobile device is on the LAN or WLAN. A machine with a lot of memory call a **Network Memory Server** [6] is located on the same LAN or WLAN and hosts the MMC on behalf of the mobile. The Network Memory Server simultaneously stores blocks belonging to the mobile node on two Persistent Storage Servers. PSS1 is located on a high-speed SAN to which the Network Memory Server is also attached. PSS2 is a persistent storage server on a Remote Storage Site and is used to provide redundancy. If local storage servers fail, the remote storage server is used to service the mobile node while local services are restored.

Let us now imagine that the mobile node is moved to another environment; the employee moves from his office at work to home where he resides. When the mobile node is turned on it recognises that it is at home, it contacts the mobile node and initiates a memory cache transfer from the Network Server to the Home Server as shown in Fig. 3. The movement of the MMC to the local area network or a network close to the mobile ensures that the system can maintain a high level of performance. However, the same persistent storage servers are used so that data can always be accessed from these two sites.

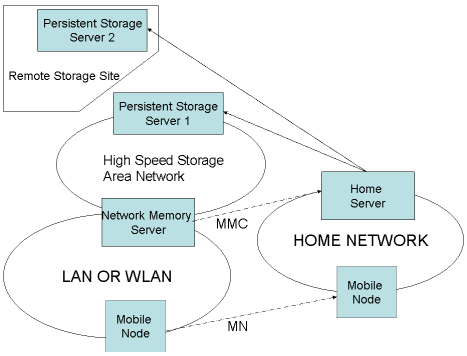

Fig. 3. Mobile Node Moves to Home Network

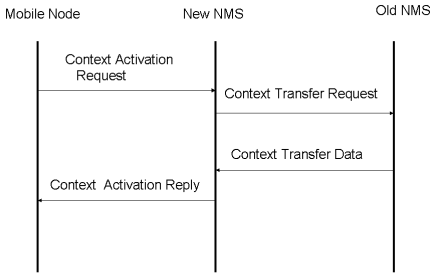

Fig. 4. Moving the Memory Cache to a Network Memory Server

1) Using the Context Transfer Protocol: In this section we show how the Context Transfer Protocol (CXTP) is used when the mobile node detects that it is on a different network compared to the location of its Mobile Memory Cache. It finds a Network Memory Server on its local network to host its MMC. The memory cache program is started on the New NMS if it is not already running. The mobile node then issues a Context Activation Message (CTAR) to the new NMS authorizing the transfer of blocks belonging to the mobile node. The new NMS sends a Context Transfer (CT) request to the old NMS requesting the data blocks belonging to the mobile node. The old NMS sends a Context Transfer Data (CTD) message to the new NMS with all the blocks belonging to the client. The new NMS sends a Context Activation Reply (CTAR) message back to the mobile node signalling that the transfer has occurred. The entire sequence is shown in Fig. 4.

2) Using Mobile IPv6 to Provide Continuous Service: With the deployment of faster wireless networks, the use of Mobile IPv6 [7] will be used to provide continuous use of the data storage system. This is because the mobility management mechanisms in Mobile IPv6 will allow the mobile node to easily detect that it has moved to a new network and initiate the Context Transfer. In this context the mobile node will always be connected to a Mobile Memory Cache so the MMC is treated as a Corresponding Node or CN. The sequence is shown in Fig. 5.

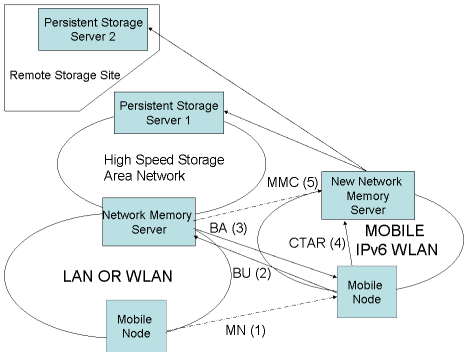

Fig. 5. Showing how continuous service is provided using Mobile IPv6

When the mobile node moves to a new network and acquires a new Care-of-Address (COA). This is shown as Step 1 in Fig 5. It sends a Binding Update (BU) to all its CNs including the MMC (Step 2). When it gets a Binding Acknowledgement (BA) it checks to see whether it is from its current MMC (Step 3). If this is so it finds a new network memory server on the new network and sends a Context Activation Message (CTAR) to the new MMC (Step 4) which results in the blocks of the mobile node being transferred to the new MMC (Step 5) using the Context Transfer protocol described above.

C. The Design of the Mobile Memory Cache (MMC)

The MMC has been designed with the following core properties: Firstly, the mobile nodes viewpoint, all actions performed by its MMC are atomic. This means that calls to the MMC either succeed completely or fail completely. Secondly, the MMC operates in a stateless manner. It does not keep track of former requests from the mobile node. Hence, it is the software on the mobile node which must maintain relevant state information. It is assumed that the MMC works over a reliable transport protocol. At present TCP/IP is used.

All references returned by the MMC should be regarded as immutable and must not be changed. The security tag in the BlockID structure will be used to detect when a BlockID has been modified. Such design decisions made the implementation of the MMC quite simple. The core operations supported by the MMC were creating and deleting ClientIDs, creating and deleting data blocks, and reading from and writing to data blocks using BlockIDs.

D. The Design of the Persistent Storage Server (PSS)

The design of the Persistent Storage Server is similar to the MMC. However, MMCs are the only clients of a PSS, its services are not directly available to mobile nodes. All operations performed by a PSS on behalf of a client should be regarded as atomic by the client. The PSS operates in a stateless manner and so keeps no record of previous interactions. It is also assumed that the PSS runs over a reliable transport link. Thus the operations of a PSS are to create and delete clients as well as to store, retrieve and delete BlockIDs. However, while the MMC uses the memory on a network server, the PSS uses disk blocks on a hard disk to provide persistent storage and so must run a file system in order to achieve this. We have therefore written a simple file server known as the **Block File System (BFS)**. The BFS takes a BlockID, a PSS ClientID and maps it to disk blocks on the hard disk. It maintains these mappings in the meta-data part of the system.

IV. Current Work

This section examines the current work being done to implement the storage infrastructure.

A. The Middlesex Testbed

In order to investigate the design of the architecture detailed above a small testbed was set up. It consists of 4 Linux machines connected on a 1Gbps network. A prototype MMC was designed and tested using the machines in the testbed. The mobile node interacted with the MMC using a pseudo network device. This allowed the blocks on the MMC to appear as a fast hard disk to the mobile node. The OS on the mobile node was totally unaware that the MMC was being used. Every call to read and write file blocks was passed to the pseudo driver which went over the network to the MMC to store or retrieve the data. At present, we are experimenting with an MMC that can cache around 200 MBs of data. In the near future we want to support an MMC with 4 GBs of cached storage.

In order to test the system, an ext2 filing system was run on the mobile node and Iozone [8], a well-known file system benchmark, was used to test the performance of the MMC. Iozone was also run on the local hard disk in order to compare the results. The disk had a 60 GB capacity, a 2 MB internal buffer, spinning at 7200 rpm with a seek latency of 9 ms and an average latency of 4.1 ms.

For our tests we created a 200 MB partition and built an ext2 filing system on it. The size of a file block was set to the default value of 1024 bytes. The network memory server was an Intel processor-based Linux machine, running at 2.4211 GHz supporting 2 virtual CPUs using Hyper-threading. It had a cache size of 512 KBs and a total memory of 512 MBs. The system uses TCP and standard UNIX socket programming. Both the pseudo device driver and MMC were written as kernel modules which can be dynamically loaded.

V. Preliminary Results

Some preliminary results using the Iozone Benchmark is given below.

A. Read Results

Fig. 6 shows the read results using different file sizes during the IOzone test. In MMC-RAW, each request to read and write blocks is immediately sent over the network. The results for MMC-RAW and the hard disk suggest that the new architecture is able to give comparable read performance.

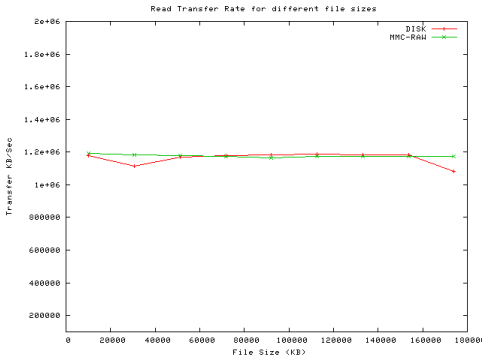

Fig. 6. Showing the Read Results between MMC and the Hard Disk

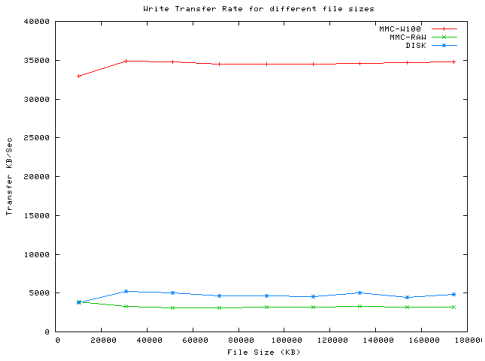

Fig. 7. Showing the Write Results between MCC and the Hard Disk

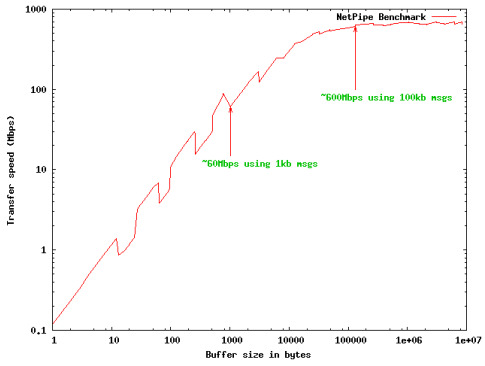

Fig. 8. TCP Netpipe Results for different Send Buffer Sizes

B. Write Results

The write results are shown in Fig. 7. Here the MMC-RAW underperforms the hard disk. An investigation into the phenomenon was done and a TCP Netpipe [9] test was performed on the Gigabit Network. The results, shown in 8, indicate that only writing a block of data (1024 bytes or 1KB) does not provide significant network performance (only 60 Mbps).

In order to improve the write results a 100 KB buffer was used to store several blocks on the mobile node. Instead of writing each block back across the network, it is first written to the write buffer. When the buffer is full, the data is then written back to the MMC. In addition a write-back thread is employed which writes back if the buffer is not full so that the data on the MMC is kept up-to-date. The result, denoted by MMC-W100 in Fig. 7, shows a significant performance improvement over the hard disk and MMC-RAW.

Work is also being carried out to provide an analytical model of network memory servers [10]. This will help the understanding of some of the relevant performance issues which are still to be addressed.

VI. Related Work

Distributed Storage systems have been developed and built over many years. Perhaps the best known work in this area is OceanStore [11], which looked at providing ubiquitous global storage based on the use of untrusted servers. Hence the data had to be protected through redundancy and cryptographic techniques. It also looked at usage patterns and adaptive algorithms to cope with regional outages and denial-of-service attacks. OceanStore pre-dated the rise of the Mobile Internet with wireless infrastructure and mobile devices so the thrust of that effort was different compared with this effort. In addition, the support of mobility using the migration of services via the Context Transfer Protocol and Mobile IPv6 allows the development of a high-performance, ubiquitous system to be developed whereas OceanStore scalability is limited. Another major effort was the Serverless Network Filing System [12] or xFS, which looked at using a set of machines in a peer-to-peer fashion to provide storage for other machines over a wide-area network. Like the Andrew File System (AFS) [13], the xFS design attempted to make extensive use of both memory caches and local on-disk cache at the client nodes and used clustering as well as sophisticated cache coherency algorithms to eliminate the need for a central server at the core of the system. Since our mobile devices do not have a local disk, some of the work is not relevant, however as we are making use of large write buffers on the mobile client, some of the cache coherency protocols are very relevant to this work and will be adopted. The SAMSON (Scalable Active Memory Server on a Network) project [14] built a Network Memory Server specifically to do remote paging based on Myrinet hardware [15]. This involved interfacing to the virtual memory system of the client and using memory mapped techniques to page in required pages as page faults occurred. Like most projects in this area, SAMSON used special hardware which provide lower network to obtain good performance. We eschew the use of special hardware as we believe that it is important to attempt to build the system using off-the-shelf components in order to reduce cost. This paper has showed that commodity hardware with local buffering aided by sophisticated caching algorithms as used in xFS is probably the best way of building a commercial system. Finally, the use of the Context Transfer protocol to migrate lightweight services in order to support mobile devices is increasing. It was used in [16] to support a Virtual Network Computing (VNC) environment [17] as mobile nodes were used to view a remote screen using VNC techniques. As the mobile mode moves to a new network, a VNC proxy server is also migrated to the new network in

order to provide sustainable performance. We believe the use of such techniques is an effective way to support the provision of services for mobile clients.

VII. Conclusions and Future Work

In this paper the design of a storage system for mobile devices was outlined. The system uses a two-level approach consisting of Mobile Memory Caches and Persistent Storage Servers. The Mobile Memory Cache uses Mobile IPv6 and the Context Transfer Protocol (CXTP) to ensure that it is in close proximity to the mobile node as it moves around. A testbed was built to evaluate the performance of the MMC. It was shown that the introduction of a write buffer greatly improves the write performance of the MMC, guaranteeing sustainable performance. This overcomes the need for special hardware which was extensively used in previous systems. We are debugging our implementation of the Persistent Storage Server and are in the process of evaluating the performance of the system over a wireless network. Work has also begun to specify a new multi-service transport protocol called X4 to replace TCP which can better support wireless environments with much lower latency. In addition since data is going over wide-area networks support for very fast encryption and decryption algorithms will be required. Multicast transport services must also be included. The authors recognize that there is much to do and so would welcome feedback on this paper.

References

[1] G. Mapp, D. Cottingham, F. Shaikh, P. Vidales, L. Patanapongpibul, J. Balioisian, and J. Crowcroft, "An architectural framework for heterogeneous networking," in *The International Conference on Wireless Information Networks and Systems*, August 2006, pp. 5–12.

[2] "A definition of 802.11 technologies," http://en.wikipedia.org/wiki/IEEE_802.11.

[3] E. Felton and J. Zahorjan, "Issues in the implementation of a remote memory paging system," University of Washington, Tech. Rep., March 1991.

[4] P. Radkov, L. Yin, P. Goyal, P. Sarkar, and P. Shenoy, "A performance comparision of nfs and iscsi for ip-networked storage," in *The 3rd Usenix Conference on File and Storage Technologies (FAST 2004)*, April 2004.

[5] J. Loughney, M. Nakhjiri, C. Perkins, , and R. Koodli, *RFC 4067 - Context Transfer Protocol (CXTP)*, IETF, July 2005.

[6] G. Mapp, D. Thakkar, and D. Silcott, "Network memory servers: An idea whose time has come," in *Multi-Service Networks (MSN)*, Coseners House, Abington, UK, July 2004.

[7] D. Johnson, C. Perkins, and J. Arkko, *RFC 3775 - Mobility Support in IPv6.*, IETF, June 2004.

[8] "The iozone filesystem benchmark," http://www.iozone.org.

[9] "Netpipe," http://www.scl.ameslab.gov/netpipe.

[10] O. Gemikonakli, G. Mapp, D. Thakker, and E. E, "Modelling and performability analysis of network memory servers," in *39th Annual Simulation Symposium*, April 2006.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and Z. B, "Oceanstore: An architecture for global-scale persistent storage," in *Proceedings of ACM ASPLOS*, November 2000.

[12] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang, "Serverless network file systems," in *15th Symposium on Operating Systems Principles, ACM Transactions on Computer Systems*, 1995.

[13] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West, "Scale and performance in a distributed file system," *ACM Transactions on Computer Systems*, 1988.

[14] "The samson project," http://bsd7.cs.sunysb.edu/ samson/design-1999/overview.html#overview.

[15] "An overview of myrinet," http://www.myri.com/myrinet/overview.

[16] L. Patanapongpibul, G. Mapp, and J. Hopper, "An end-system approach to mobility management for 4g networks and its application to thin-client computing." in *ACM SIGMOBILE Mobile Computing and Communications Review*, July 2006.

[17] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, pp. 33–38, January 1998.