CriStore: Dynamic Storage System for Heterogeneous Devices in Off-site Ubiquitous Communities

Hyunbin Lee YongJoo Song Kyungbaek Kim Donggook Kim Daeyeon Park
Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology (KAIST)
373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Korea
{hblee, yjsong, kbkim, dgkim}@sslab.kaist.ac.kr, daeyeon@ee.kaist.ac.kr

ABSTRACT

Most researches for ubiquitous services have been interested in constructing intelligent environments in physical spaces such as conference hall, meeting room, home, and campus. In these spaces people are able to share data easily. However, as cooperative works from a distance make a social issue, we need a data sharing system for an "*off-site*" community that is a group of people with a common interest and purpose in different ubiquitous places like a remote conference/meeting. In this paper, we propose a dynamic storage system, *CriStore* for heterogenous devices of the off-site communities, which autonomously builds a distributed shared data space and keeps a flexible overlay topology of the participant devices according to the devices' capabilities. CriStore also performs file operations fitted to the capabilities.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management—*distributed file systems*

General Terms

Design

Keywords

Dynamic Storage System, Off-site Ubiquitous Community, Heterogeneity, Self-organizing, Flexible Overlay Topology

1. INTRODUCTION

Most of ubiquitous computing researches so far, focus on constructing next generation computing environments [2, 4] in geographic regions with limited and well-defined physical boundaries like home [11], meeting room [12], conference hall [10], and campus. The ubiquitous environment is composed of an intelligent space containing appliances (video projectors, lighting sensor, etc), powerful stationary servers, and wireless handheld/wearable devices. Users interact with the space, which provides various services such as data sharing and location detection. In these ubiquitous surroundings, the mobile devices are increasing, both in number and diversity [1, 14] as hardware becomes more compact and more powerful by the evolving technologies and designed for diverse applications. The heterogeneity of devices stands for different capabilities in terms of processor performance, memory and storage capacity, display size, network connectivity, and battery power. Thus there are not only strong devices like laptops and desktop computers holding plentiful resources but also weak devices such as PDAs, smart phones, and so on.

Having the manifold devices in the same space, a group of people with a common interest or a specific purpose is able to share the data via the ubiquitous environment. However, people who are located in different ubiquitous places or where there are no ubiquitous environments (except wired/wireless networks) may also desire to share the distributed data each other with only their devices: for example, remote conference/meeting among members in variant regions, research data sharing among students in different universities, and multimedia file sharing among people who live in diverse areas. We call the latter group an "*off-site*" ubiquitous community from now on. The off-site community is generally temporal and heterogeneous. The community is suddenly organized for aims like above the examples and broken up after completing its cooperative work. The devices of the participants are various and someone can use multiple devices. The off-site community with these two characteristics requires that a storage system must be able to autonomously build a distributed shared data space without specific static servers and keep a flexible overlay topology depending on the heterogeneity of the devices. We propose a file system layer approach so that every ubiquitous framework and application makes use of the storage system without special libraries and recompiling.

In this paper, we propose CriStore, a dynamic storage system for the data sharing and adapted streaming in the off-site ubiquitous community with the various devices. To simply construct a shared data space in the dynamic manner, CriStore self-organizes an overlay topology of the devices without the static file servers. Moreover our system recognizes the heterogeneity of the participant devices including processor performance, memory/storage capacity, network connectivity and battery power. Each CriStore device acts as a server or a client according to the device's capabilities.

SAC'07 March 11-15, 2007, Seoul, Korea

The powerful devices construct a "*ring server group*" with a ring-shaped network topology, and this ring server group provides the data sharing and the adapted streaming services to the weak devices working as the clients. The server group manages the whole shared data for the off-site community and guarantees both of availability and consistency. The clients on the feeble devices just access the remote data by using streaming protocols with a content adaptation for fitting the original data to capabilities and preferences of the devices.

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 outlines the design of CriStore. We describe the detailed implementation in Section 4. Finally, Section 5 concludes this paper.

2. RELATED WORK

In order to understand the challenges posed by the autonomy and the heterogeneity of the off-site ubiquitous community, let us begin by considering the previous distributed file systems. The existing systems are divided into two categories: one is the system with static file servers such as Coda [5, 6] and AFS [7, 8]. The other is the system without the static servers such as xFS [13] and Segank [9].

The system with the static servers consists of client-only nodes and server-only nodes. All shared data is stored on the dedicated static server nodes which provide the shared data space. If a client node wants to share the data with other ones, it should store the data at the server node. Thus if a new off-site community is formed, it must establish the static server nodes to accept and accommodate the client nodes. Especially in the case of Coda and AFS mostly used by the existing ubiquitous frameworks, a skilled administrator is required to build the systems and manage the server nodes. It is not autonomous for general users to establish the static servers whenever they form the community temporally. The static server approaches have another problem in the heterogeneity. Since Coda and AFS assume that the client node is a strong device like a personal computer and a powerful laptop, the client node caches a entire file read from the static server node in its local storage for later accesses with less network traffic. It is the high overhead to the weak client node as the cached file size becomes large. While the client device is caching a large file like a multimedia file, applications concurrently running on the device may be blocked by the burst writes to the storage. Moreover the device with the low computing capabilities would be hard to instantly uncompress the multimedia file encoded by a compact codec like DivX, AVI and MPEG. If the device has the small storage capacity, it can not even cache the file in its local storage.

The serverless approaches use the wholly distributed architecture. Since the peer devices communicate with each other directly, the serverless system autonomously constructs the shared data space with no dedicated static servers. xFS is comprised of the client-mode devices and the server-mode devices without the static servers. xFS clusters the high capable devices into a server group and operates based on the cooperated cache: it uses all main memory of the powerful server-mode devices to share the data in a fast LAN. However, the network of the off-site communities is not the local area but non-uniform wide area where diverse network links exist. It is the critical problem that the data access of the devices using xFS in the WAN may be delayed by the networks.

Segank is devised for storage elements connected by the heterogeneous networks unlike xFS. The Segank users are able to aggregate data blocks of a desired file from a number of widely distributed devices using a multicast tree called Segankast. However it is large overload to construct the multicast tree and gather the dispersed data blocks whenever reading the file. The weak devices with the low computing capabilities are hard to run the Segank system. Furthermore, as designed for a personal storage, Segank is not suitable to the multi-user environment of the off-site community. For the consistency of the data accessed by the personal user, Segank requires each user to carry a small device, MOAD which records the sequence of write operations.

3. SYSTEM DESIGN

We design the dynamic storage system for the off-site ubiquitous community with the heterogeneous devices, CriStore that performs proper operations fitted to the devices' capabilities. When a device enters into the off-site community, it is classified into a server of the ring server group or a client according to its computing capabilities, battery power, and network connectivity. The powerful devices constructs the server group as a core system to provide the distributed shared data space to the weak devices without the static servers. In Figure 1 the personal computer A, B and the power supplied laptop C with the high processor clock and the large memory/storage capacity organize the ring server group with the optimal network connectivity between the servers. The mobile device Z with the low capabilities acts as the client. The operating systems are able to detect these hardware capabilities via bios interfaces and device drivers. CriStore stores a configuration file including personal preferences as well as the client's capabilities: the client's capabilities and the personal preferences are abbreviated to "*CC/PP*" from this time.

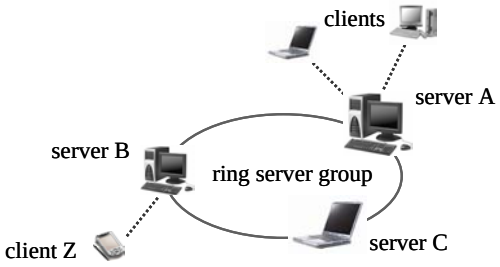

Figure 1: CriStore Overlay Topology

The devices of the ring server group can autonomously organize their ring-shaped network topology by themselves. Even though traditional distributed systems exploit existing structured peer-to-peer topologies [3, 16] for scalability, we select the simple ring topology because we assume that the size of the off-site community is not large: it consists of dozens of the various devices. Moreover file systems using the peer-to-peer like Ivy [15] have the critical problem of high overhead due to aggregating data blocks of a file from modification logs whenever reading the file.

To provide the transparent data sharing, this ring server group maintains all shared file information including name, location and version. The file name and location like an

IP address are used to search the shared file among many servers. The file version composed of a version count and an IP address is used to check the data consistency between the devices. To optimize and keep the ring topology, the ring server group also maintains locations of all servers and every network link cost between the neighbor servers via ping operations.

Each CriStore device should know at least one IP address as a "*connection point*" in order to access to the ring server group. The device requests a new connection to a "*target server*" among the connection points. Diverse applications on CriStore do general file operations such as reading multimedia files and writing text files. CriStore accesses the shared file by the streaming protocol based on a block read request protocol, and the accessed files become adapted to the device's capabilities. The device can cache the file read from the ring server group or discard the file depending on its capacity or demand.

4. IMPLEMENTATION

4.1 System Overview

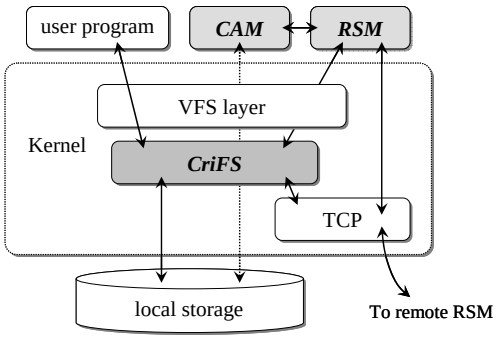

Figure 2: CriStore Architecture

Figure 2 shows our CriStore architecture. CriStore comprises three modules which are core kernel module, ring support module, and content adaptation module. The core kernel module, CriFS as a file system basically handles VFS operations like open, read, and write and mediates shared data accesses. It communicates with the local and remote ring support modules for the file operations and selectively caches the accessed data in its local storage. CriFS also detects the hardware capabilities and records them in the CC/PP file. To simplify our implementation, we perform both the ring support module and the content adaptation module in application-level daemon. The ring support module, RSM recognizes its own capabilities from the CC/PP, and self-organizes the ring server group or acts as the client. RSM also supports the availability and the consistency of the shared data among every device and recovers the ring topology resiliently when a server crashes down. The content adaptation module, CAM in the ring server group transforms the original data into another one adequate for the device's capabilities. More detail operations and mechanisms are described in the following section.

4.2 Self-organizing Ring Server Group

Every RSM of the ring server group maintains the ring topology by keeping each connection with RSMs of immediate fore and back neighbor servers. In Figure 3 counterclockwise servers of the ring imply the fore neighbors and clockwise servers imply the back neighbors. RSM of the servers has current ring topology information in order that a new joining server finds an optimal network connectivity and rearranges the ring topology. The ring topology information contains IP addresses of all the servers and every network link cost from the neighbor servers. Server-side RSM periodically measures round trip time to the neighbor servers using the ping operations, and transmits the recorded network state information to all the servers via the ring or piggybacks it in sent other packets to the next server.

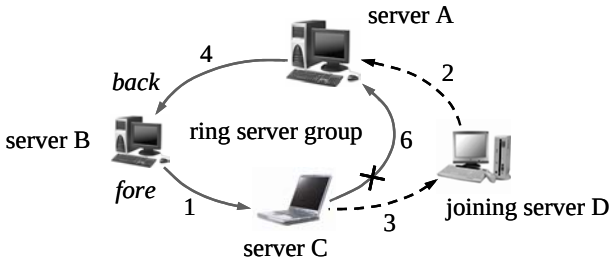

Figure 3: Join Process to the Ring Server Group

Figure 3 illustrates the sequence of a join process to the ring server group. The numbers in the figure represent the network link cost between two servers. If a device (D) as a server joins a target server via the already known connection points, the target server notifies the ring topology information to the device. The device compares all the previous network link costs with the new link cost, and it selects a new target server for the optimal ring topology as the following algorithm. Because the network link between the servers is identical with a backbone of the entire systems, the ring server group optimally reconstructs its topology.

Algorithm 4.1: SELECT_NEW_TARGET_SERVER(all_link_costs)

```
// i means the i^th server
// (x, y) stands for the network link cost between x and y

for (from i = 0 to n, i++) {
      if (((new device, i) + (new device, i+1)) − (i, i+1) ≤ 0) {
            return the i^th server
      }
}
return the random server
```

If the new target server (A) is determined by the algorithm, the server accepts the joining device as a new back neighbor (D) and transmits a "*rejoin*" message to the previous back neighbor (C) to update the ring server group. Receiving the rejoin message which has the IP address of the new device (D), the previous back neighbor (C) breaks up the prior connection of the fore neighbor server (A) and requests the new connection to the joining device (D) as a new fore neighbor (D). After completing the new ring topology, the joining device checks cache consistency with the ring server group. The device sends its own shared file information via the ring and receives the shared file information of the ring server group from the target server. A leave process from the server group is equal to the reverse of the join process.

4.3 File Operations

The difference between the file operations on the client and those on the server is whether the CriStore device accesses the shared file via the target server or not. In the case of the server, CriFS directly accesses the file using the shared file information without help of the target server. In this section we describe the file operations on the client.

CriStore does the file-based operations like Coda and AFS in order to reduce complexity of the operations and the amount of network traffic. When opening the shared file accessed by the applications, CriFS recognizes whether the file exists in the local storage or the remote ring server group. If the applications want to read the file that is not cached in the local storage, CriFS requests the file to the ring server group via the target server using a block-based remote read protocol that is designed for a random play on a streaming service. In the beginning of the remote file request, CriFS sends the CC/PP information once to the target server. CAM of the server uses this CC/PP information to transform the original file into another one adequate for the client's capabilities. CriFS can also selectively cache the file received from the ring server group based on the client's capabilities or demand. On the other hand if the shared file required by the applications is cached in the local storage on the opening time, CriFS not only transmits the CC/PP information but also queries the version information of the original file in the target server in order to compare the version of the local file with that of the remote file. If the file in the local storage is obsolete, CriFS discards the old file and reads the updated file again from the target server. If the target server has no file required by the client, it searches the closest server having the file using the shared file information and the ping operations, and relays the file to the client. At this "*relay read operation*", the target server caches the file taken from other server and it acts as a proxy server for the sake of temporal locality. Similarly the client changes its connection point to new one where the file frequently accessed by it exists after several relay read operations for the proximity between the client and the server.

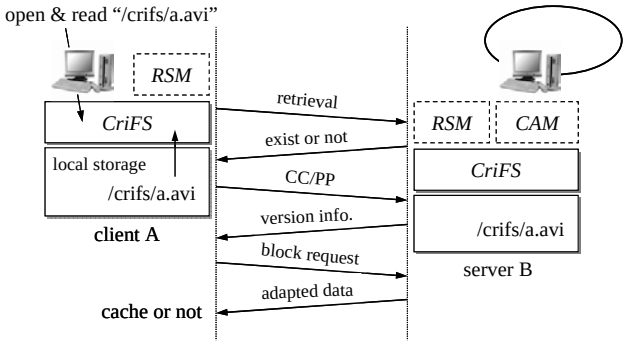

Figure 4: Read Operation

On writing a new file created by the applications, CriFS stores the file not only in the target server but also in the local storage if the client is capable to cache the file. If the applications update the file, CriFS sends modified data blocks of the file to upper-layer RSM via a socket in order to transmit the modified data blocks to the target server. RSM queues modification logs in its memory, which mean aggregated file operation logs containing the modified data blocks. It transmits all the modification logs to the target server when the log size exceeds threshold bytes or the predetermined time is expired. This burst transmission reduces the energy consumption at the network interface of the mobile devices and helps the devices to work on an even disconnected network link. Using the logs passed to the ring server group via the target server, each server updates the modified file and its version information. This operation is more and less lazy, but it is not the crucial problem because the members of the off-site community are not large and generally the write operations are still smaller than the read operations.

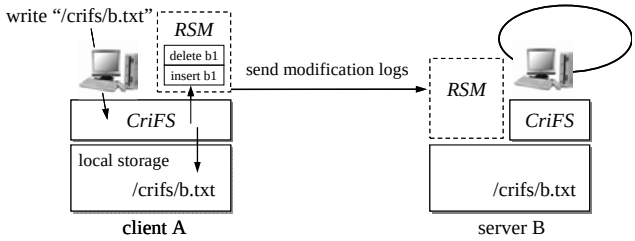

Figure 5: Write Operation

4.4 Cache Management

Reading the shared file from the remote server, the conventional distributed file systems such as Coda and AFS cache the entire file in their local storage by all means. The reason is that the systems exploit the cached file on the disconnected network link and reduce the network traffic occurred whenever reading the same file from the server. However, reading the file from the server and caching the file in the local storage at the same time may impose the high processing overhead on the device and causes a spatial problem if the device has not enough space of the storage. Thus CriFS makes the decision whether it caches the file received from the server or not according to the device's capabilities. In addition because the users usually read the head of a file only to confirm if the file is the exact one, CriFS forks a background kernel thread starting to cache the file after the read data blocks of the file exceed the certain threshold amounts.

4.5 Cache Consistency

CriFS needs a mechanism to guarantee the cache consistency through the version in the shared file information because some CriStore devices in the off-site community are able to cache and concurrently update the same files. This version information consists of the reusable 64-bit version count and the user IP address. Whenever a file is modified, the version count increases one by one for representing the recent one. If several users of the community attempt to update the distributed files concurrently, the updated files have the same version count but the different user IP address in the version information. In this collision case, CriStore creates another file with a new file name which has with a postfix ".ver[i]" at the end of the original name except the file type (e.g., sample.ver[1].txt). The integer count i is initialized to zero and increased one by one whenever the collision between the files occurs. Later the users can select the desired one from the files having the different extended

name. This approach has the lower latency than the approach using a lock on the writing time like a token ring.

4.6 Fault Tolerance

Every member of the ring server group knows the entire ring configuration (n "*connection points*"of the fore servers counterclockwise within the ring) based on the ring information. If a neighbor server crashes, RSM detects the disconnection and then reconnects to the nearest one among the fore servers. After completion of reconstructing the new ring topology, RSM updates the shared file information and checks its version information to keep the cache consistency among the servers. On a client crash, CriStrore has only to be just rebooted and check the cache consistency because all shared data that the client has exist and are managed in the ring server group.

4.7 Content Adaptation

The content adaptation is a kind of stream service that transforms the original data to another one fitted to the device's capabilities and commonly applied to a read-only or multimedia file in the server. When a device tries to access the shared file, CAM of the server handling the request executes the content adaptation based on the device's CC/PP information. CAM determines the stream quality according to the CC/PP, and provides the adapted file to the device. CAM also stores the transformed file in the local storage of the server so that the server sends the stored file to other devices without the transformation overhead when the devices having the similar capabilities request the same file later.

5. CONCLUSION

In this paper we propose a dynamic storage system that is designed for heterogenous devices of "*off-site*" ubiquitous communities. *CriStore* autonomously constructs a distributed shared data space without static servers. Every applications on CriStore is able to perform general file operations and transparently access shared files any where. CriStore also supports a flexible network topology of the devices according to their capabilities. More powerful devices self-organize a "*ring server group*" as a core system, which provides a stream service fitted to the device's capabilities by a content adaptation. Moreover CriStore with a selective cache management guarantees cache consistency and fault tolerance. Furthermore, if CriStore device acting as the server exhausts its battery power, the devise can dynamically convert server mode to client mode. CriStore consequently considers an energy-efficiency as well as a performance of the entire system.

6. REFERENCES

[1] K. T. Krishnakumar and M. Sloman. Constraint-Based Network Adaptation for Ubiquitous Applications. In *Proceedings of the Sixth IEEE International Enterprise Distributed Object Computing Conference (EDOC'02)*, pp. 258–271.

[2] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. In *IEEE Pervasive Computing*, Vol. 1, No. 4, pp. 74–83, Oct.–Dec. 2002.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 ACM SIGCOMM Conference (SIGCOMM'01)*, pp. 161–172.

[4] J. P. Sousa and D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In *Proceedings of the Third Working IEEE/IFIP Conference on Software Architecture (WICSA'02)*, pp. 29–43.

[5] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. In *IEEE Transactions on Computers*, Vol. 39, No. 4, pp. 447–459, April 1990.

[6] M. Satyanarayanan. The Evolution of Coda. In *ACM Transactions on Computer Systems (TOCS'02)*, Vol. 20, No. 2, pp. 85–124, May 2002.

[7] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, et al. Scale and Performance in a Distributed File System. In *ACM Transactions on Computer Systems (TOCS'88)*, Vol. 6, No. 1, pp. 51–81, Feb. 1988.

[8] M. Satyanarayanan. Scalable, Secure, and Highly Available Distributed File Access. In *IEEE Transactions on Computers*, Vol. 23, No. 5, pp. 9–18, 20–21, May 1990.

[9] S. Sobti, N. Garg, F. Zheng, J. Lai, Y. Shao, C. Zhang, E. Ziskind, et al. Segank: A Distributed Mobile Storage System. In *Proceedings of the Third Annual USENIX Conference on File and Storage Technologies (FAST'04)*, pp. 85-124.

[10] J. Barton and T. Kindberg. The CoolTown User Experience. In *HP Labs Technical Report HPL-2001-22*, available as http://www.hpl.hp.com/techreports/2001/HPL-2001-22.html

[11] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving: Technologies for Intelligent Environments. In *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing (HUC'00)*, pp. 12–29.

[12] P. Tandler. Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices. In *Proceedings of the Third International Conference on Ubiquitous Computing (Ubicomp'01)*, pp. 96–115.

[13] T. E. Anderson, M. D. Dahlin, J. M. Neefe, et al. Serverless Network File Systems. In *ACM Transactions on Computer Systems (TOCS'96)*, Vol. 14, No. 1, pp. 41–79, Feb. 1996.

[14] A. Karypidis and S. Lalis. OmniStore: A System for Ubiquitous Personal Storage Management. In *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'06)*, pp. 136–147.

[15] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A Read/Write Peer-to-Peer File System. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, pp. 31–44.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference (SIGCOMM'01)*, pp. 149–160.