Peer-to-Peer File Systems

Seminar of Distributed Computing

DCG ETH Zürich

Hannes Geissbuehler (hannesg@student.ethz.ch)

November 2003

1 Summary of Ivy, a peer-to-peer file system

The paper describing Ivy [ivy] introduces a multi user read/write peer-to-peer file system with NFS-like semantics. Ivy was developed at the MIT and is based on a DHash and a Chord layer which both were also developed at MIT and published in the papers [cfs] and [chord].

The Ivy file system consists solely of a set of logs. A log is a linked list of immutable log records where each log record describes a change on the file system. Ivy's logs contain version vectors that allow it to detect and resolve conflicts. Each user has it's own log-head which points to the most recent own log record. The log-head of each user is signed with it's private key to provide authentication. When many participants want to build a file system together, they have to agree on a corporate view. A view block therefore contains pointers to all log-heads and the root Inode. To get access to other users log, users must exchange their public keys. From time to time each participant constructs or updates it's private snapshot. A snapshot is the actual state of the distributed file system on which one take part. While building this snapshots file and directory Inodes will be produced out of the log records and stored in the system.

To store log records, snapshots and Inodes, Ivy uses DHash which is the software layer directly underneath Ivy. DHash was first developed as part of the CFS read only storage system (in this system only the creator can change and create files, all other users have read access only). DHash take files, splits them into blocks (in order of tens of kB) and distributes them using the underlying Chord layer. To assign a key to each block DHash hashes the entire content of each block except for log-heads for which DHash uses the public key of the block as key. DHash is a kind of service layer therefore it implements replication of blocks, caching of blocks during lookup and load balance through the hash function and the use of virtual servers. DHash provides no explicit delete operation but stores data only during a finite time interval.

On the bottom of the layers we have Chord distributing and locating the blocks over all available nodes. Chord is a distributed lookup protocol, which is designed to efficiently locate a particular node in a large dynamic peer-to-peer system with frequent node arrivals and departures. Basically, Chord provides support for just one single operation: given a key, Chord maps it onto a specific node (in fact it returns the IP-address of the node). Chord uses a variant of consistent hashing to assign each node an 160-bit identifier by hashing the node's IP address and virtual server ID. A key identifier is produced by hashing the key itself. The identifiers are ordered in an identifier circle on which every key K is assigned to the first node whose identifier is equal to or follows k on the circle. This node is called successor node. Every node has a list of its r nearest successor nodes and additional routing information stored in a so called finger table to handle queries in a efficient way. This design results in a simple, robust and good scalable protocol with provable correctness and provable performance.

2 Analysis

Now as we have with Ivy an implemented peer-to-peer read/write peer-to-peer system following question arises: are the problems of Ivy negligible in practical use and do we really need such a file system and if yes who will use it then? In my eyes, Ivy has three major problems, which are not negligible: waste of disk space, security issues and unresolvable conflicts.

- Waste of disk space:

  In all 3 papers they just assume that disk space is cheap therefore we have enough disk space. This assumption may be not so wrong, disk space has become in fact very cheap, but to waste

it and use it as a write once data media is just unacceptable. The problem is: in Ivy you cannot modify existing blocks. Every time you change only one little thing you have to create a completely new block and you can't even delete the old one, because you have to store each log forever. This storing is necessary because in case of conflicts you may need old log records. And even if you wanted to delete one of the block you couldn't because no delete operation is supported. The only way to delete something is not do refresh the time interval for which DHash stores certain block. This indirect delete operation brings new troubles along: one need something like a daemon, which updates your blocks from time to time. This is very time consuming and error prone. A solution might be to implement a delete operation or to provide something like a garbage collector for unused blocks.

- Security issues:

  Because Ivy is based on Chord and DHash it naturally inherits all problems of this two layers. And I see a big security problem in the fact, that Chord trusts in every node. Therefore a few malicious participants running many virtual servers can present an incorrect view of the Chord ring. And an incorrect Chord ring brings along conflicts in the DHash and Ivy layers. I think in a system which is designed for frequent node arrivals and departures (for example mobile devices) the security mechanism should already be implemented on the Chord layer and not only in the Ivy layer. So one solution could be to use public key authentification already on the Chord layer or that every node has to register itself first. Maybe in the future one could use the security function of TCPA to authentificate each node in a easy way.

- Unresolvable conflicts:

  In Ivy it is possible that unresolvable conflicts can occur (for example if the underlying Chord ring is partitioned). For this case, Ivy provides tools to manually repair the conflicts. I think every system-administrator or manager who is in charge to decide which system is to be used will instantly decide against Ivy when he first reads about unresolvable conflicts. One solution might be to cut down the file system semantics that no more unresolvable conflicts can occur. Having banned unresolvable conflicts we would have solved also a big part of the disk space problem because we then could delete unused old blocks.

Including this three problems we have to discuss where Ivy could be used. One big area for Ivy could be to replace servers and backups system or distributed network file system like the AFS. In this domain I see other problems for Ivy. First we have the three mayor problems of Ivy discussed before, which are in most situation counterarguments enough. Then server systems are nowadays not that expensive anymore and one knows more ore less how to administrate them. On the other side, Ivy means that you have to install an Ivy instance on every client and that you are using experimental infrastructure. Not to forget the performance handicap of Ivy (Ivy is two to three times slower than NFS).

In the years of Napster, Kazaa and E-donkey one should consider Ivy as a future file sharing tool. Although Ivy scales very good I think Ivy is inoperative for file-sharing purpose. First every user must have the public-keys of every other user he wants to share with, then users don't want to have replicas on their computer. The need of a powerful search makes it necessary to have up to date snapshots. Thus means that we have to go trough the logs of all participants what is not feasible in the big file-sharing community. This shows that traditional file-sharing clients have other requirements than to have full file system semantics.

To summarize, I think that although Ivy is already implemented and has interesting new ideas it will never be used in a commercial way. I see Ivy as a typically scientific implementation to test new ideas and collecting experience. Fixing the actual problems, peer-to-peer file system eventually will become more important in the future, when data has to be available everywhere in every time. Also the growing sector of embedded systems and sensor networks is a potential application area. They are highly connected but have small memory so they could store their data over a peer-to-peer file system.

References

[ivy] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, Benjie Chen: *Ivy: A Read/Write Peer-to-Peer File System*;

[cfs] Frank Dabek, Frans Kaashoek, David Karger, Robert Morris, Ion Stoica: *Wide-area cooperative storage with CFS*; SOSP'01

[chord] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan : *A Scalable Peer-to-peer Lookup Service for Internet Applications*; SIGCOMM'01