Keso, A scalable, reliable and secure read/write peer-to-peer file system

Keso was developed as a master thesis project at KTH/IMIT by
Johanna Svenningsson <mea@kth.se> and Mattias Amnefelt <mattiasa@kth.se>.
The work was supervised by Luc Onana Alima <onana@sics.se>

Design goals

Keso is designed to be a distributed file system which is run without dedicated servers and where the participating machines each store a part of the data.

The main design criteria are that Keso should

- Make use of unutilized resources - there is a lot of free space on workstations today
- Avoid storing redundant data - a lot of data is stored multiple times in file systems
- Scale well - should support thousands of clients
- Be self organizing - no administrative intervention when nodes join or leave
- Be a secure file system suitable for a real world environment

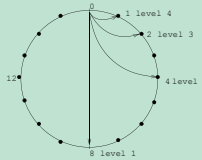
DKS

We use DKS as the foundation for Keso. DKS

- is a structured overlay network developed at Sics and KTH.
- scales well and is self organizing.
- supports a high level of dynamism
- provides a distributed hash table to Keso.

Data and nodes are given identifiers. These identifiers are mapped onto a ring. Data belongs to the closest node which follows numerically in clockwise order on the ring.

Each node has pointers to nodes at exponentially increasing distance. A node only needs to maintain information about a small part of the system.



Data can be found, inserted and deleted using a number of messages logarithmic to the number of nodes in the system.

Implementation

A prototype implementation is available in C++. It can set up a network between nodes. Directories can be created, and files can be added and retrieved from the file system. As nodes join and leave the system data is migrated between nodes.

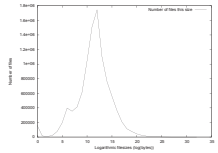
The implementation contains a command line client which the user uses to interface with the client. There is currently no kernel interface. The implementation is prepared to be able to use the NNPFS kernel module from the Arla project.

The implementation does not currently support the designed security mechanism. This work is planned for the near future.

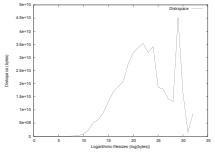
A brief look at a real file system

A study was made of the distributed computer system at the IT-Department of KTH. Our measurements show that 24% of all data stored in the central file system is redundant and that 50% of the hard drive space of workstations is unused.

The department run an AFS-Cell with approximately 500 GB of data. We studied both the distribution of files as relating to the size of the files and the contents of the files to discover if there was redundant data in the file system. This was done by computing a SHA1 hash of each 256K block of the file system and then comparing the hashes, and if two blocks had the same hash we compared the actual data. There were no collisions found in the hash function.



The number of files relating to the size of the files.



The amount of data in files of a certain size.

This shows that most files are small but most data is stored in a few large files. The information from these measurements has been taken into account when designing the Keso file system. A file system which avoid storing unneccesery redundant data will save a lot of disk space.

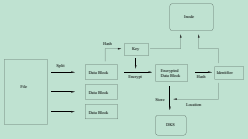
The file system

Files are split into blocks, which are stored in the Distributed Hash Table. Inodes represent files. They contain metadata and a block list. Blocks are referenced from this block list. Each time a file is changed, a new inode is created, but only the disk blocks which have been changed are written. Each unique data block is only stored once and is referenced from all files which contain that block. This way no redundant data is stored in the file system.

A directory is a set of change records for each file. When a file is deleted, a special Delete marker is inserted into the list of change records. This gives us the ability to retrieve older versions of files.

Security

Disk blocks are encrypted. In order to make it possible to save redundant space, a hash is calculated from the content of the block. The block is then encrypted using a symmetric crypto with this hash as the key. Another hash is then computed using the encrypted content of the block. This hash is used as the location when storing the block in the Distributed Hash Table.



Both the hash of the clear text block and the encrypted block are stored in the block list of the inode. The inode is then encrypted with a key from the directory.

A directory contains an ACL which lists the right of each user with access to the directory. Since each file is encrypted with a symmetric key, we need to limit access to this key to protect the file. This can be done by encrypting the key using the public keys of all users with access to the file. This ensures that only users who have the right to read a file can do so.

When a user writes to a directory, then the user also creates a signature over the changes made. This signature is verified by the node which stores the directory. The signature is also stored in the directory. This ensures that we always can verity the validity of a directory at each point in time.

The name Keso

Inspiration for the name Keso originally comes from the Arla project. Arla is a free client for the Andrew File System. The Arla project has always named things with dairy related names. Some examples are the AFS client Arla, which is a major Swedish dairy distributor, the file server Milko which is another dairy distributor and the kernel module NNPFS, which is named after a cooperative union of dairy producers called NNP.

The name Keso follows these lines and is a bad pun in Swedish.