The IGOR File System for Efficient Data Distribution in the GRID

Kendy Kutzner and Thomas Fuhrmann

University of Karlsruhe (TH), Germany
`(kutzner|fuhrmann)@ira.uka.de`

Abstract

Many GRID applications such as drug discovery in the pharmaceutical industry or simulations in meteorology and generally in the earth sciences rely on large data bases. Historically, these data bases are flat files on the order of several hundred megabytes each. Today, sites often need to download dozens or hundreds of such files before they can start a simulation or analysis run, even if the respective application accesses only small fractions of the respective files.

The IGOR file system (which has been developed within the EU FP6 SIMDAT project), addresses the need for an easy and efficient way to access large files across the Internet. IGOR-FS is especially suited for (potentially globally) distributed sites that read or modify only small portions of the files. IGOR-FS provides fine grained versioning and backup capabilities; and it is built on strong cryptography to protect confidential data both in the network and on the local sites storage systems.

1 Introduction and Motivation

One aspect of grid computing is the access to remote data. These remote data resources can be organised in files of arbitrary size. Especially the access to large remote files is challenging if these files change frequently. For example in grid applications in the pharmaceutical industry the size of such collections of files often exceed 2^{39} bytes. The data volume grows fast, since life science is a prospering industry. This challenge for data storage and data transport will not be solved by Moore's Law, because the growth rate of the data volume is higher than the growth rate of bandwidth and storage capacities. Since the data may be non-public and is transported over the Internet, authentication and confidentiality are required. Current solutions are not optimal because they often transfer files even if only a small part of the file has been changed. Further, current solutions put high loads on the server machines if many clients access simultaneously. Storage capacities and network capabilities are pushed.

This paper proposed a solution based on overlay network technology. Such decentralised and distributed overlay networks allow the sharing of resources, so the burden on the central server is relieved. Since data security (confidentiality and authentication) are key factors for grid applications, our solution especially address this area. To ease the deployment, we separate the networking components from the file storage part of our proposal. Because we use the normal

POSIX file system as interface, all applications immediately are able to use our new file system.

The paper consists of three main parts. First we define our problem and give an outline of our solution. Second, the IGOR network is described in section 2 as an essential substrate for the IGOR file system. Such a decentralised and distributed overlay network allows the easy sharing of resources like storage capacity and bandwidth. The IGOR file system itself is thirdly explained in section 3. There the security aspects of our solution are detailed in this section as well as the the easy integration with all applications because of the use of POSIX as interface between the IGOR file system and applications.

2 IGOR overlay network

The IGOR overlay network is a key based routing system. It was inspired by systems like Chord [9] and Kademlia [8]. Overlay network have the unique ability to shape their own topology in a way that for example eases key based routing. Key based routing systems are the foundation of many diverse applications like multicast, publish/subscribe systems, message exchange, data storage, voice transfer, distributed hash tables and many others. The main concept of key based routing systems and therefore of IGOR as well are

- All nodes act as routers as well as end systems. That means nodes forward messages for other nodes.
- Messages are targeted to identifiers, usually from a large identifier space.
- Nodes bear identifiers from the same identifier space.
- Messages are forwarded through nodes until they arrive at the node which identifier is closest to the destination identifier of the message. One difference between different key based routing schemes is the metric they use to measure closeness between identifiers.

One of the important property of IGOR is that it can deliver messages in $O(\log N)$ hops (intermediate forwarding systems) while maintaining connections to $O(\log N)$ other nodes (N being the number of nodes in the network). As such the IGOR overlay network is very scalable.

Message forwarding in IGOR does not only consider the destination identifier of the message, every message also carries an application or service identifier. With this identifier, IGOR delivers messages only to nodes where an application with the given identifier is registered. This allows that multiple applications run over a shared overlay network. Currently a distributed video recorder[7], a test suite[5] and a SIP-proxy[2] have been implemented besides IGOR-FS on top of the IGOR-API.

A node running IGOR should usually be placed in a DMZ, so it easily can connect to other IGOR-nodes. Applications within the firewalled region can connect to IGOR and use the IGOR-API to talk to remote nodes. However IGOR also implements the ability to traverse some types of network address translation (NAT) system, so it can also be deployed inside organisations. Further IGOR features proximity neighbour selection (PNS, [4]), so the topology of the IGOR network will adapt to the physical network. Also PRS (proximity route selection [4]) is used to utilise the overlay network in a very efficient way.

The file system described in this paper is based on IGOR and inherits all the benefits from it. The separation of network related aspects from file system related aspects allow the deployment of IGOR and IGOR-FS nodes on different machines for example within the DMZ or as a trusted computer within the firewalled network.

3 IGOR File System

IGOR-FS is based on the IGOR overlay network, a Chord[9] variant which combines peer-to-peer technology and service orientation. As such, it has similarities with systems like [1] and [6]. With IGOR, all participating machines automatically organise into a structured virtual overlay network. Resources and services are addressed via hashed application keys. Furthermore, IGOR is proximity and quality of service aware, that is, it prefers a responsive or local resource over an unresponsive or remote resource, both when building the overlay network (neighbour selection) and when routing requests (route selection)[4].

The IGOR file system uses the FUSE technology [3] to be mounted into the Linux file system name space. Thereby, it converts all file system structures such as folders, links and the files itself on the fly into variable size blocks, which can be easily sent across the IGOR network. If a mounting site knows both the block identifier and the decryption key of a file block or a folder it has read access. If it authenticates as source site for that file or folder, it has write access, too. Reading and writing may affect only a small fraction of the entire file system. Hence the network traffic is greatly reduced. Additionally, IGOR-FS can store several versions of such blocks and thereby efficiently reproduce several consistent versions of the entire file system. Moreover, sites may cache the individual blocks to speed up further access to the same part of the data. This eliminates bottleneck problems that are typical for centralised data distribution approaches.

3.1 File System in User Space

The file system described in this work is based on FUSE[3]. Because of that, application can continue to use the usual POSIX file system interface, i. e. IGOR-FS can be used by applications without the need to change any of them. FUSE exposes the internal virtual file system (VFS) layer to the user space. This has the advantage that IGOR-FS can be implemented entirely in user space. That offers better debugging support during development, better library support and easier use of other user space features like BSD-sockets. FUSE allows mounting and un-mounting of file systems by ordinary users, i. e. no elevated user privileges are required besides the initial enabling of FUSE.

3.2 Block Encryption

Files should not be transported as a single entity. First this would create large latencies, because a request can not be served until the position of that file has been transfered. A second reason not to transfer entire files is that probably only a small part of the file is needed at the requesting side. For this reason IGOR-FS splits files in blocks. Unlike file systems on regular hard disks, the sector size of 512 octets is of no relevance. Blocks can be of arbitrary sizes, but have to fulfil the following requirements:

1. Block size must be reasonable. Blocks that are too large take too much time to transfer and thereby generate unnecessary latencies. If users want only parts of that block, other parts are transfered without any need. On the other hand, if block sizes are too small, a large overhead is created because every block transfer requires some signalling. The optimum depends on expected usage patterns, expected bandwidth and the overhead to transport one block.
2. Equivalent pieces in different files should be detected, i. e. they should be chopped the same way.
3. Equivalent files should be cut in the same way at different hosts without the need for additional communication. In fact such a signalling may not be possible if the cutting happens off-line. That means the algorithm should run deterministically.
4. Small modifications in a file should influence the cutting algorithm only locally. That means when the content of one block changes, most block boundaries should stay the same.
5. Insertion or deletion of some data should influence block boundaries as slightly as possible.

Our approach is to compute the block boundaries by a rolling checksum. Taken the file as continuous data stream of octets d_i, the rolling check sum with bit width b at position x is computed as

$$\mathrm{RC}_x = \bigoplus_{k=0}^{b-1} d_{x-k} << k \quad \mathrm{mod} 2^k$$

With this definition, the continuous (rolling) checksum is easy to compute. Given the checksum at one position x, the checksum at position $x+1$ follows $\mathrm{RC}_{x+1} = (\mathrm{RC}_x << 1) \oplus d_{x+1}$, i. e. no previous data is required. If RC_x falls below a threshold, a cutting mark is inserted. To adapt the algorithm to the desired block size the threshold can be adjusted. Block cutting is followed by an optimisation process to avoid very large and very small blocks.

After a file has been cut into blocks, every block is encrypted as follows. First the block is hashed with cryptographic hash function $H(B) = k$. The result of the hash function is yields key for encryption $E_k(B) = E_{H(b)}(B)$ of the block. After encryption, the block is hashed again to gain an identifier of block $ID(B) = H(E_{H(b)}(B))$. This scheme has the advantages that a piece of data is encrypted in the same way on different hosts. Blocks transporting a block can check the integrity of the encrypted block by recomputing $ID(B)$. However the scheme still ensures that the knowledge of the encrypted block and the block identifier does not allow a node to decrypt data passing by. Only possession of block ID $ID(B)$ and decryption key k allows fetching, decryption and again verification of the block. Note that every data block is encrypted with a different key, therefore the ability to read one block does not imply the readability of any other block.

3.3 Directory Structure

As explained in the previous section, individual data blocks are encrypted separately. This section will describe how files are made of single blocks and how directories describe multiple files. Every single block is decryptable and therefore readable if the pair (ID, k) for that block is known. A list of (offset, ID, k)-tuples form a file. Such a serialised list is again stored in a block. If this block becomes large, it will be subject to the rules for block cutting as described above. It is always subject to the same encryption scheme, which will result in a single (ID, k)-pair for every file.

Directories associate such a pair with a file name and other attributes of files like creation and modification time and Unix file permissions. Soft links are also stored in directories as an association between two strings, link name and link target. Further objects stored in directories are other directories. After all objects in a directory are collected, the directory content (basically (name, ID, k)-tuples and soft links) is serialised. The result of the serialisation process is again put into the block cutting and encryption process as described in section 3.2, which will result in a block identifier ID and an encryption/decryption key k. Such a pair can be either put in a parent directory, or can be transfered to any other party. When anybody learns such an (ID, k) describing a directory block, he can read and decrypt this directory. Since the directory contains block identifiers and decryption keys for files and sub-directories, these can be fetched and read too. Thus a single block identifier (plus its corresponding key) allows to read a whole directory structure. That also implies that only these two pieces of information are required to be transfered over a outside channel. Note that using this single directory block identifier gives a constant view of the directory tree below it. Changing any single file gives a new top level directory identifier. On the other hand, using any previous directory identifier, gives previous states of the tree. That opens many possibilities for backup, rollback and versioning.

3.4 Block Transfer

The block transfer in IGOR-FS happens on demand only. That means that if no node is reading date, no data block has to be transported through the network. To identify block during requests and transfers, the block ID $ID(B)$ as described in section 3.2 is used. The property of the used hash function ensures that every node in the network and especially the requestor can verify the integrity of the block. The authenticity of block identifiers (and therefore blocks) is ensured by identifiers of the parent directories. For the topmost block identifier, the outside channel is responsible for authenticity and integrity.

The block identifier is used as the key in the key based routing network to look up nodes in possession of the requested block. As soon as a block is retrieved, the retrieving node becomes another source for this block, i. e. every node may cache any block. Because of the encryption of the blocks content, the confidentiality is not at risk. PRS ensures that from many possible cached copies the copy close by in the physical network is selected. Because of the caching in the network, the more popular a given block becomes, the faster the access to this block will be. This effectively avoids bottlenecks at hot spots.

Because the block cutting algorithm tries to find similar blocks in files, a modification in a file probably alters only few blocks. Requesters of the new file which already have a cached copy of the old file therefore only have to transfer the changed blocks. Large files with frequent small changes as given in our scenario benefit a lot from this mechanism.

4 Summary and Conclusions

We presented IGOR-FS, an overlay network based file system able to share large files between different organisations. It is well suited for frequently changing files that need to be transfered authenticated and confidential. IGOR-FS enables the secure sharing of network and storage capacities among participants.

The implementation of IGOR-FS is complete and the evaluation is currently underway. The prototype is available for experimental use. It has been successfully employed with large bio-informatics data bases within the SIMDAT project.

5 Acknowledgements

This work is funded under EU grant number 511438.

References

1. Ian Clarke, Theodore W. Hong, Scott G. Miller, Oskar Sandberg, and Brandon Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.
2. Johannes Eickhold. Entwicklung einer SSR-basierten Peer-to-Peer-Telefonieanwendung für das Nokia 770, December 2006.
3. File systems in user space. http://fuse.sourceforge.net.
4. K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the SIGCOMM 2003 conference*, pages 381–394. ACM Press, 2003.
5. Ivan Kostov. Developing an automated test environment for the overlay network IGOR, December 2006.
6. John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris

Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 190–201. ACM Press, 2000.

7. Kendy Kutzner, Curt Cramer, and Thomas Fuhrmann. A self-organizing job scheduling algorithm for a distributed vdr. In *Workshop "Peer-to-Peer-Systeme und -Anwendungen", 14. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2005)*, Kaiserslautern, Germany, February 2005.
8. Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer-Verlag, 2002.
9. Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Technical report, MIT Laboratory for Computer Science, PDOS Group, 2002.