Improving DHT-Based File System Performance with B-trees

Róbert Dávid and Gábor Vincze

In this paper, we present a way of improving the performance of network file systems based on Distributed Hash Tables (DHT) by incorporating B-tree variant into the file system.

System Structure

The proposed system has four subsystem layers:

- File System (user interface, top layer)
- Search trees, indexing
- Reliable data block storage
- DHT (data block location and routing)

The top layer is a conventional file system interface: it should show to users files, attributes, directories, and should provide tools to add, modify, copy, delete files and their metadata. DHTs offer a scalable and completely decentralized storage mechanism [1, 2, 3, 4]. However, DHTs only solve the problem of Decentralized Object Location and Routing (DOLR), and do not provide any means for indexing and searching contents of the network. Many solutions have been proposed to build an indexing layer on top of the DHT, and offer file system functionality [5, 6]. In the Cooperative File System, a DHash layer provides redundancy and caching of file system data and metadata blocks, on top of which a File System layer provides the file system interface to users. In our system, we replace the simple replication of DHash with a more efficient block-coding algorithm, like IDA [7] or Reed- Solomon block coding. In our system, we add a search tree/indexing layer between the reliable data block storage and the File System. Modern conventional file systems use B-trees to improve performance [8], and outperform non-B-tree file systems by a factor of 10-15x in certain usage scenarios. We hope to use B-trees in DHT-based file systems to reduce the number of routing steps necessary to locate data blocks, which can have a significant impact in the case of a globally distributed network.

Search Trees

The search tree we propose is a B+-tree, we call C-tree, C for content addressing. Taking advantage of content addressing, we can increase the branching factor, while node sizes in bytes can stay the same. The concept is the following: Let the address of tree-nodes be calculated from the first and last index value, one can reach using the node in question. I proposed a hash function to spread nodes data uniformly over the DHT network. Now child nodes can be addressed without a pointer, we need only the "lowerthan- node" and "higher-than-node" index values the B+ algorithm already stores, and use the hash function. A little downside is that we must store the low and high limit of the node, but it is out weighted by the benefit of not having to store node addresses implicitly. An extra advantage is that the root blocks address will be always well known, because it must be the hash of the index of all zero bits and all one bits. To install multiple file systems over one DHT can be guaranteed to use of keyed cryptographic hash functions, encrypted storage, user specific keys.

Conclusion

By incorporating B-trees into DHT-based file systems, we hope to take DHT-based file systems beyond the simple initial architectures used in CFS or Pastis.

References

[1] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications, In *Proc. ACM SIGCOMM* (San Diego, aug. 2001).

[2] S.P. Ratnasamy. A scalable content-addressable network (2002)

[3] P. Druschel and A. Rowstron. Past: Persistent and anonymus storage in a peer-to-peer networking environment, In *Proceedings of the 8th IEEE workshop on hot topics in operating systemy* (HOTOS 2001), (Elmau/Oberbayern, Germany, may 2001), PP. 65-70.

[4] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. TAPESTRY: An infrastructure for fault-tolerant wide area location routing, (2001)

[5] F. Dabek. A cooperative file system, (2001)

[6] J.-M. Busca, F. Picconi, and P. Sens. PASTIS: A highly-scalable multi-user peer-to-peer file system, (2004)

[7] M.O. Rabin. Efficient dispersal of information for security, Load balancing, and fault tolerance, *JACM*, Vol. 36, no. 2, april 1989.

[8] REISERFS4, `http://www.namesys.com/V4/V4.html`