CREATIVE MATH. & INF. **Online version at** http://creative-mathematics.ubm.ro/
17 (2008), No. 3, 363 - 368 **Print Edition: ISSN 1584 - 286X Online Edition: ISSN 1843 - 441X**

Dedicated to Professor Iulian Coroian on the occasion of his 70^{th} *anniversary*

BBUFs: Synchronization mechanism

DAN COJOCAR

ABSTRACT. Peer-to-peer systems are distributed systems without any centralized control or hierarchical organization. All nodes from a system have identical capabilities and responsibilities, and all communications are symmetric. BBUFs (Babeş Bolyai University File System) is a peer-to-peer file system. The file system is designed for ordinary Unix machines that are IPV6 capable. In this paper we present our approach regarding the synchronization mechanism.

1. INTRODUCTION

Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization. All nodes from a system have identical capabilities and responsibilities, and all communications are symmetric. The advantage of using peer-to-peer systems comes from the capabilities of these nodes that are unused most of the time. The disadvantage of peer-to-peer systems is that most of the nodes in the system are transient, and may be available only for short period of time. That is why such systems should provide means for dealing with networks nodes leaving and joining repeatedly. For peer-to-peer distributed file systems this means that replica of files / directories should be maintained on multiple nodes [12]. Replication of files / directories (or data) may achieve many goals [4]:

- *High-performance*. By replication a file / directory at different locations (geographically), the access time may be minimized and the load on the communication network may be decreased [1].
- *Greater availability*. If different physical copies of a file / directory are distributed in the system, a replicated resource is more available than a single-copy [9].
- *Fault tolerance*. A failure that occurs to one physical replica of a file / directory does not compromise the access to the logical file / directory since other replicas exist in the system [8].

However, the presence of replica files / directories within a distributed file system also introduces one important problem: *replica consistency*: if the status of a physical replica changes (through *write* operations), all other replicas of the same file / directory must reflect this change. The way this problem is handled in a distributed file system is called *synchronization mechanism*.

Received: 29.10.2008. In revised form: 03.04.2009. Accepted: 14.05.2009.
2000 *Mathematics Subject Classification.* 68M12, 68M14.
Key words and phrases. *Synchronization, replication, distributed file system, ipv6, peer-to-peer.*

The rest of the paper is structured as follows. Section 2 presents some of the synchronization mechanisms used, so far, for file replication. Section 3 introduces BBUFs. In Section 4 we present the synchronization mechanism used in BBUFs file system and the advantages and disadvantages of this solution. Some conclusions and further work are given in Section 5.

2. Related Work

Sandhu and Zhou proposed a scheme for cluster-based file replication in large-scale distributed file systems [13]. A cluster is a group of workstations and one or more file servers on a local area network. Large distributed file systems may have tens or hundreds of clusters connected by a backbone network. By dynamically creating and maintaining replicas of shared files on the file servers in the clusters using those files, they reduce reliance on central servers supporting such files and they reduce the distance between the accessing sites and data.

Page et al. have proposed a file replication mechanism via *stackable layers* [11]. They used a stackable layers architecture to permit replication to coexist with other features. Also, their system, called Ficus, supports a high degree of availability for *write* operations by allowing updates during network partitions provided at least on replica is accessible.

Cabri et al. have proposed an adaptive approach for file replication [4]. They integrated into the distributed file system an adaptive file replication policy that is capable of reacting to changes in the patterns of access to the file system by dynamically creating or deleting replicas.

Chen et al. proposed a fault-tolerant model for replication in distributed file systems [5]. They use the ideas of directory-oriented replication and the extended prefix table with an incorporated two-level heap data structure.

Plavec and Czajkwski proposed a distributed file replication system based on distributed hash tables [12]. Each node in their system acts as a client, issuing data requests and, also, as a storage node, storing a portion of the system data. The nodes can join and leave the system at any time, without warning. The system adapts to the changes in the system by migrating among nodes and generating new replicas to keep the number of replicas in the system constant. They use a versioning consistence protocol to handle data changes.

3. BBUFs

BBUFs (Babeş Bolyai University File System) is a peer-to-peer decentralized file system that has similar characteristics to Ivy [10] and Pastis [3]:

- *highly scalable* - benefit of decentralization of peer-to-peer system;
- *fault tolerant* - a file can be configured to have n number of replicas on different nodes;
- *highly performable* - each client ”talks” directly to the nearest node that is providing the requested content.

But BBUFs differentiates itself from these file systems because is not using a structure like Chord or DHash to build or store the overlay network of the file system [2]. We are using a new mechanism based on IPV6 addressing scheme

[7]. Also, each node has no knowledge about other nodes except how to contact a node from his group.

In our approach a group is a set of nodes that are replicating the same directory. This means that a node can be part of many such groups. The group is represented by:

- an IPV6 address that will be allocated from the subnetwork dedicated to anycast use;
- a multicast IPV6 address that will be allocated from the multicast subnetwork corresponding to the anycast address defined above.

The BBUFs system is using three types of addresses:

- *unicast* - used for data communication between clients and nodes of the file system;
- *anycast* - used for lookup messages between clients and system nodes;
- *multicast* - used between the system nodes (i.e., when synchronizing their shared content).

In our system each node will have at least one address from each of the above defined group:

- *an unicast address* - representing this node on our file system network. This address is assigned by the file system administrator.
- *at least one anycast address* - representing at least one directory shared by this node. This address is computed automatically using the BBUFsMapper algorithm, described in [6].
- *at least one multicast address* - representing the multicast group that the current node is registered on. This address is also computed by the BBUFsMapper algorithm.

4. PROPOSAL

In order to make BBUFs a scalable, reliable and fault tolerant peer to peer distributed file system, we have to take into consideration the file replication problem. In order to solve this problem, we have to deal with the following issues:

- How do we synchronize the shared content?
- Where to create a new replica?
- How to spot changes easily?
- Which node has the last version?

4.1. **SyncDaemon.**

BBUFs *SyncDaemon* is a program that is responsible for maintaining data synchronized between nodes. Periodically *SyncDaemon* performs the following tasks:

- Contact all the nodes in its *multicast group* to check for replicas health.
- Call health check routines to verify local shared data.
- It determines when to create a new replica.
- It initiates a synchronization process when it spots differences of content.

Using the *multicast group* we are able to ”talk” only with the nodes that are sharing this content. This way we are not overloading our system with broadcast messages that recipient nodes are not interested.

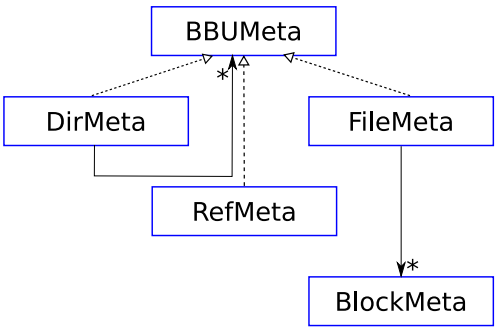

FIGURE 1. Metadata class diagram.

SyncDaemon will maintain for each shared content a structure named *BBUMeta* that will be the metadata representing the entire directory structure.

BBUMeta, as presented in Figure 1, is the base class for:

- *DirMeta* - the metadata object representing directories that are shared by our file system.
- *FileMeta* - metadata for representing a file that is shared by our system.
- *RefMeta* - a reference to an entry that is stored on another node.

BBUMeta and all derived objects are containing the following information:

- *name* - the name of the represented object.
- *n* - the minimum number of desired replicas that the system will try to maintain.
- *hash* - the SHA1 hash value of the shared object.

DirMeta also contains an array of *BBUMeta*. For each entry of a directory we will have a *BBUMeta* object.

Each *FileMeta* object will represent a file of the shared content with the following attributes:

- *version* - the version of this entry.
- *size* - the size in bytes of this entry.
- *BlockMeta* - the metadata used for representing the file blocks.

BlockMeta represents the actual information contained in a file divided in blocks. The block size will be established by the file system administrator, and for each such a block we will maintain the following information:

- *hash* - the hash of this block.
- *version* - the version of this block.
- *BlockMeta* -the next block information or a null reference if this is the last block.

RefMeta will contain, besides the information inherited from *BBUMeta*, only a link information to another node(s). This information is just a tip for the client, because the nodes may not be available or the shared information could be relocated.

When *SyncDaemon* determines that a file is out of sync, using the information from *FileMeta* object, it will perform the following steps:

- Will determine the block(s) that has(have) changed using the hash and version information.
- If the versions are the same it will generate a conflict.
- If one of the nodes is having a block with a lower version it will append the new block(s) to the destination file. In Figure 2 we have presented this case. Here the daemon has spotted that the second block has changed and is performing an append operation and then it is rebuilding the links.
- Will update the *FileMeta* information on the destination file to reflect the new changes.

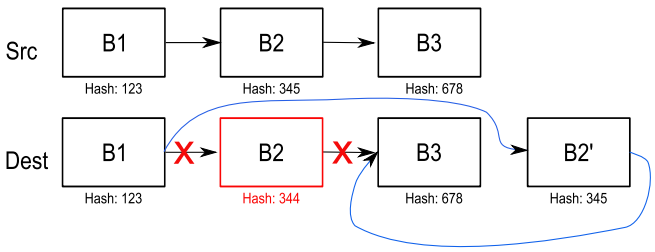

FIGURE 2. Replace block sample.

When the daemon will determine that we have a file conflict it will perform the following actions:

- Create files with .res.*date* suffix for each version, where the *date* is the actual modification time.
- Notify the file owners, by email, and the file system administrator to resolve the conflict.
- Store over the original file the version with the last modification time.

After a conflict, the owners or the administrator will be able to choose one of the two .res.*date* files to keep.

Each time a node receives a read / lookup request, it will log the request. When the *SyncDaemon* detects that for a shared directory we do not have the desired number of copies it will perform the following steps:

- Analyze the logs to determine the networks from where it has received requests for this content.
- Will determine the network that has need of such a content.
- Will send a broadcast request to determine a node of this network that will meet the requirements.
- Initiate synchronization with the first node that is responding.

4.2. **Advantages/Disadvantages.** The approach that we propose has the following advantages:

- Using small broadcast groups we are exchanging synchronization messages only to interested nodes.

- Each node is able to establish where the system needs more replicas.
- Using the *FileMetadata* each file is able to heal very quickly.
- The nodes that are out of sync are quickly spotted.

The disadvantages of our approach are:

- The need for a garbage collector.
- The files that are updated frequently will contain unused blocks.

5. Conclusions and Further Work

The BBUFs peer to peer distributed file system was briefly introduced in this paper. Its synchronization mechanism was described in depth. The advantages and disadvantages of this approach were also presented.

Further work will be done in the following directions:

- To deal with security concerns that appear in these type of systems.
- To formalize the synchronization mechanism.
- To compare the performance of our approach with other mechanisms.

References

[1] Bal, H.E., Kaashoek, M.F., Tanenbaum, A.S. and Jansen J., *Replication techniques for speeding up parallel applications on distributed systems*, Concurrency: Pract. Exper., **4** (5) 337-355, 1992

[2] Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R. and Stoica, I., *Looking up data in p2p systems*, Commun. ACM, **46** (2) 43-48, 2003

[3] Busca, J.M., Picconi, F. and Sens, P. Pastis, *A highly-scalable multi-user peer-to-peer file system*, in Euro-Par, Vol. 3648 of Lecture Notes in Computer Science (Jos C. Cunha and Pedro D. Medeiros, Eds.), pp. 1173-1182, Springer, 2005

[4] Cabri, G., Corradi, A. and Zambonelli, F., *Experience of adaptive replication in distributed file systems*, in EUROMICRO Conference, pp 459-466, Los Alamitos, CA, USA, 1996 IEEE Computer Society

[5] Chen, T.S., Chang, C.Y., Sheu, J.P. and Yu G.J., *A fault-tolerant model for replication in distributed-file systems*, in Proceedings of the National Science Council, Republic of China, Part A, Physical Science and Engineering, Vol. 23, pp. 402-410, 1999

[6] Cojocar, D., *BBUFs: A new lookup mechanism based on ipv6*, in Proceedings of GlobalComp Workshop, held in conjunction with 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 42-45, Timişoara, Romania, 2008

[7] Deering, S. and Hinden, R., *Internet protocol*, version 6 (ipv6) specification, 1998

[8] Huang, Y. and Tripathi, S. K., *Resource allocation for primary-site fault-tolerant systems*, IEEE Transactions on Software Engineering, **19** (2), 108-119, 1993

[9] Ladin, R., Liskov, B., Shrira, L. and Ghemawat, S., *Providing high availability using lazy replication*, ACM Trans. Comput. Syst., **10** (4), 360-391, 1992

[10] Muthitacharoen, A., Morris, R., Gil, T.M. and Chen, B., *Ivy: a read/write peer-to-peer file system*, SIGOPS Oper. Syst. Rev., **36**(SI): 31-44, 2002

[11] Page, T.W., Popek, G.J., Guy, R.G. and Heidemann, J.S., *The Ficus distributed file system: Replication via stackable layers. Technical Report CSD-900009*, University of California, Los Angeles, CA (USA), 1990

[12] Plavec, F. and Czajkowski, T., *Distributed File Replication System based on FreePastry DHT. Technical report*, University of Toronto, Ontario, Canada, 2004

[13] Sandhu, H.S. and Zhou, S., *Cluster-based file replication in large-scale distributed systems*, SIGMETRICS Performance Evaluation Review, **20** (1), 91-102, 1992

Babeş-Bolyai University
Department of Computer Science
1 M. Kogalniceanu Street
400084 Cluj-Napoca, Romania
E-mail address: dan@cs.ubbcluj.ro