SMALL FILE PACKING IN DISTRIBUTED PARALLEL FILE SYSTEMS - PROJECT PROPOSAL

Srinivas Chellappa and Mikhail Chainani and Faraz Shaikh

{schellap, mchainan, fshaikh}@andrew.cmu.edu
Carnegie Mellon University

ABSTRACT

Distributed parallel file systems are typically optimized for accesses to large files. Small file performance in certain parallel file systems may potentially be improved by storing small files in their entirety along with the file metadata information. This paper proposes a design to do so on PVFS2 and explores how such a design could be evaluated.

1. INTRODUCTION AND BACKGROUND

Distributed parallel file systems are file systems that distribute file data across multiple storage components to provide higher concurrent performance. These typically appear as a single, persistent system to clients. Distributed parallel file systems exist for various reasons, the primary ones being the reduction of access bottlenecks and the delivery of high performance in concurrent environments.

Many current distributed parallel systems focus on optimizing performance for large files, and either support small files at lower performance levels, or not support small files at all [1, 2]. Though this is not an issue for applications that process vast amounts of data in large chunks, applications that would benefit from storing and accessing small files (say, less than 10k) either have to live wih the reduced performance, or have to include an extra software or hardware layer to be able to eke better peformance out of the parallel file system.

The problem of low performance is exacerbated in systems like PVFS2 [3] that use separate systems for metadata and actual storage. This requires the clients to first contact a metadata system to acquire the correct storage location, and a second, separate access to the appropriate storage location to actually acquire the data or the file. Such accesses are typically bound in latency and bandwidth by local area networks like ethernet. This is not an issue in terms of bandwidth performance for accessing large files, since the cost of the two accesses is amortized over the large size of the files. However, the latency of accessing small files on such a parallel file system might be large because the overhead costs would dominate.

The goal of this project is to propose and study a design to store small files in their entirety along with the associated metadata in such distributed parallel storage systems. The hope is to reduce the costs of accessing small files by eliminating the separate access to the storage system for small files.

Paper Organization. This section has provided a brief introduction and background. Section 2 presents related work. Section 3 motivates and presents the problem statement. Section 4 presents our proposed design. Section 5 presents our proposed evaluation methods.

2. RELATED WORK

The file system that we propose to implement and evaluate our design on is PVFS2 [3]. PVFS2 is an open source parallel file system designed for use in large scale cluster computing. PVFS2 is optimized for large dataset access performance [1, 2]. Increasing the performance of PVFS2 for small file/dataset access without affecting large dataset access performance is a desired outcome of this project.

Packing and storing small files along with metadata information is not a new idea, especially in the domain of traditional local file systems. NTFS packs small files ($<$ 1500 bytes) into the metadata structure itself [4]. However, parallel file systems typically do not do this.

Some distributed parallel file systems take other approaches to optimizing small file access. MSFSS [5] is a distributed parallel file system that optimizes small file access by using an underlying file system (ReiserFS) that is optimized for small file access. However, the type of systems we are targetting are already optimized at all levels for large file access, and since changing that is not an option, this approach does not work.

Devulapalli et al. in [6] describe a method to decrease network traffic utilization during file creation in a parallel file system. A side effect of our approach is that it would achieve the same effect, thus increasing available bandwidth for large file accesses.

3. PROBLEM STATEMENT

As previously mentioned, certain distributed parallel systems like PVFS2 include separate metadata servers and storage systems. Since there is a fixed overhead associated with making multiple accesses to obtain a file, file sizes below a certain threshold might benefit from being stored on the metadata server itself. Our project proposes to implement such a design and determine such a threshold in addition to evaluating the general viability of such an approach.

4. APPROACH AND DESIGN

PVFS2 uses a database to maintain and serve metadata. Our approach is to modify PVFS2 to store files with sizes under a given threshold in the metadata server system. Below, we present an overview of PVFS2 and expected modifications.

Implementation Platform. We selected PVFS2 [3] as our implementation and evaluation platform. Apart from being an open source project, as mentioned earlier, PVFS2 focuses mainly on large file performance, and our hope is to increase small file performance in PVFS2 while leaving large file performance untouched (or increasing it).

There are four major components to the PVFS system:

- Metadata server (mgr)
- I/O server (iod)
- PVFS native API (libpvfs)
- PVFS Linux kernel support

The daemon servers run on the cluster nodes. The metadata server (mgr), manages all the metadata for the files including the file name, directory heirarchy, and information about how the file is distributed across servers. The metadata is stored in a database on the mgr. Our proposed design would pack the file data along with its metadata at the mgr for small files.

The second daemon is the I/O server (iod). the I/O servers manage the data storage and retrieval from local disks. The nodes run a local file system which is leveraged by these servers. No modifications are expected to be required on the iod.

Clients can access the PVFS filesystem using two methods - via a user-space library or through Linux kernel support. In either case, the client has to be modified appropriately to support small file packing.

PVFS does not cache meta data due to complexity of maintaining consistency in a parallel access environment. Thus for small files, reads and writes will force the client to make at least two network accesses: one to the meta data server (mgr) and the other to the IO server (iod). Figure 1 and Figure 2 illustrate file access in PVFS. Our design is aimed at eliminating the access shown in Figure 2.

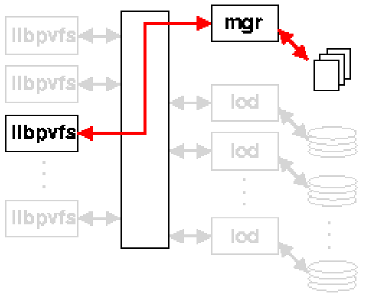

Fig. 1. A PVFS client first accesses the mgr (metadata server) (From [7])

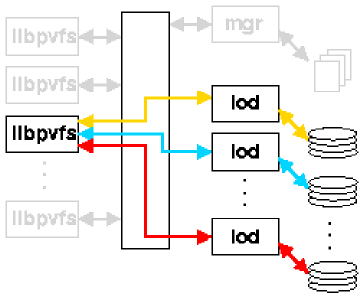

Fig. 2. A PVFS client uses information from the mgr to access the iod (actual file data store) (From [7])

Issues. There are also several issues that we will attempt to answer during the course of this project. Metadata servers might be replicated for performance, and our design might cause undesired replication of small files along with metadata replication, thus increasing storage requirements. Some distributed file systems [8] allow clients to cache meatadata, and this has to be handled appropriately.

5. EVALUATION

Evaluation System. We will evaluate our implementation on a simple setup consisting of three machines - a client, an iod (I/O server), and an mgr (metadata server), running on 100MBps ethernet. If available, we will also evaluate our implementation on larger deployments of PVFS2.

Evaluation Metrics. Our project primarily targets increasing the performance of small file accesses in distributed file systems. Hence, our primary metric will compare the latency of single accesses, and the sustained throughput of several small file accesses on our system, with the baseline. The baseline will be an unmodified PVFS2 setup.

A major goal of this project is to determine a threshold or a range of file sizes for which file packing increases performance. Measuring the access latencies and bandwidths of a range of file sizes would provide an idea of such a threshold. Note that such a threshold or range would depend on the actual distributed parallel file system under consideration, and other configuration parameters of an instance (eg.: network latencies, bandwidths).

Next, we will compare the overall performance of an actual workload across our modified and baseline systems. We will choose an appropriate set of workloads, preferably providing various mixtures of small and large file accesses.

Secondary metrics that we will use, time permitting, are the extra load on the metadata servers and the network utilization of the connections to these servers, for specific workloads. Though client access performance is ultimately what matters, these secondary metrics might provide insight into the costs and bottlenecks of our design.

6. REFERENCES

[1] R. Latham, N. Miller, R. Ross, and P. Carns, “A next generation parallel file system for linux clusters,” *LinuxWorld*, 2004.

[2] R. Latham, N. Miller, R. Ross, and P. Carns, “Shared parallel filesystems in heterogeneous linux multi-cluster environments,” *proceedings of the 6th LCI International Conference on Linux Clusters: The HPC Revolution*, 2004.

[3] The parallel virtual file system, version 2. [Online]. Available: http://www.pvfs.org/pvfs2

[4] NTFS.COM. Ntfs. [Online]. Available: http://www.ntfs.com/ntfs-mft.htm

[5] L. Yu, G. Chen, W. Wang, and J. Dong, “Msfss: A storage system for mass small files,” in *Proceedings of the 2007 11th International Conference on Computer Supported Cooperative Work in Design*, 2007.

[6] P. Devulapalli, A.; Ohio, “File creation strategies in a distributed metadata file system,” in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007.

[7] The parallel virtual file system. [Online]. Available: http://www.parl.clemson.edu/pvfs/desc.html

[8] S. Ghemawat, H. Gobioff, and S. Leung, “The google file system,” in *ACM SOSP*, Oct. 2003.