# A framework for distributed storage

Jean Lorchat
Internet Initiative Japan
Email: jean@iijlab.net

October 7, 2009

## 1 Introduction

Recent trends in consumer grade computing and entertainment, as well as technological breakthrough with respect to digital contents, have led users to own a large amount of data, without proper backup means or storage management (e.g. how can a user increase his storage capacity easily ?).

Indeed, current available products either hardware or software, rely on a number of assumptions such as the availability of a high bandwidth local area network dedicated to storage devices [1, 2], or even purchasing expensive integrated solution devices.

We wanted to introduce a framework that allows users to manage all their data by themselves at all time, provided they own all the servers participating in the architecture. This framework can be used by single individuals as well as large Internet service providers, since it is very scalable with respect to number of users.

## 2 Modular three layers architecture

In order to create a such framework, it seems beneficial to split data and meta-data information. Then, different entities handle each type of information at different layers. The entity responsible for storing file data is called a disk server, while the meta server handles meta-data from classical UNIX-like attributes to user defined tags and categories.

In addition to meta-data information, meta servers also handle storage-wide area mapping, which allows them to tell to connecting clients where the resources they are looking for is located. This is especially important because files will be split into fixed size blocks, which provide a vey fine granularity for operations such as replication and parallel I/O.

In this situation, lots of scenarios can be imagined, that combine all kind of pieces of information extracted from different contexts to achieve the features being wished for by the users of the storage system. For example, we can envision implementations that favor latency, or performance, or load-sharing, or even redundancy.

In an opposite fashion, the disk server is merely a large disk array under the supervision control of a single computer that must be able to address simple read and write requests from client entities, or from meta servers moving data around to achieve their higher-level storage strategy.

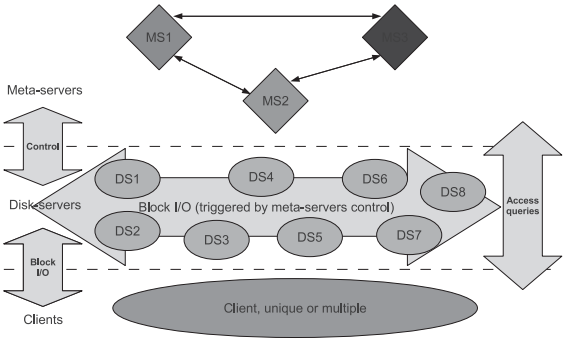


Figure 1: Framework architecture diagram

Consequently, the last entity is the client. A client is any kind of application, either modified to access the framework or created with the framework in mind. Since the storage system is not a real filesystem, it does not provide POSIX compliant semantics that would allow a seamless mounting within the user filesystem tree. It is thus necessary to resort to a separate API that is different from the usual filesystem API, even networked ones. This is caused by the highly distributed nature of the storage system, and in order to achieve the highest possible performance with a such setting. By adding a filesystem compatible abstraction on top of the framework, we also introduce a large potentially useless and wasteful overhead, impairing performance.

These three entities are pictured in Figure.1 with their relationships and the involved protocols.

## 3 Query mechanism

When the client application needs to access data, it must first obtain a connection to one of the meta servers. Only these servers are able to locate the data among the various disk servers.

After initiating the connection, the client tells the server which resource it is looking for. After checking the resource, the meta server decides which of the possible locations should be returned, based on all kind of policies. Such policies can be used to filter out overloaded disk servers, increase reliability, and so on.

## 4 Conclusion

This architecture framework is still very much a work in progress and has a lot of room for improvement, while lacking features at the same time. Our future works related to this architecture is to prepare a proof of concept implementation that can be used to validate the system.

Once the prototype is ready, meta server plugins should be written to evaluate different scenarios and see if the framework can really adapt to various kind of situations.

Eventually, among the highly wanted upcoming features within the framework, we can cite access control, privacy, data consistency and asynchronous operation.

But the key feature that motivated to start this work has been the ability to create a distributed storage covering a very wide area, as opposed to site-wide products.

## References

[1] S. Shepler, M. Eisler, and D. Noveck, “NFS version 4 minor version 1,” Dec 2008.

[2] Danga Interactive, “MogileFS,” Available online at http://www.danga.com/mogilefs/.