S3FS in the wide area

Alvaro Llanos E
M.Eng Candidate – Cornell University – Spring/2009

ABSTRACT

S3FS[1] is an interface implementation using FUSE and Amazon Storage Services in order to provide a reliable and secure storage location with a easy to use interface for Unix users. Although S3FS[1] presents a simple and functional implementation, there are several issues, inherent to WANs(Wide Area Network), not considered during its implementation.

This project explains some of the limitations of the current S3FS[1] implementation and addresses the performance issues with a simple solution based on existing designs and algorithms. Due to the "high latency" of the WANs, this project will implement an asynchronous Caching mechanism along with parallel processing of the Reads/Writes requests to S3, therefore mitigating the performance loss caused by the latency of the wide area networks.

1.1. INTRODUCTION

Wide area networks have become very important and necessary the last decade. Therefore most of the server-client applications must be able to support WANs in order to become a feasible option for most companies. The reach of Internet, the most famous and important WAN these days, makes it a valuable asset for all kind of businesses. The services offered by companies like AWS (Amazon Web Services) give smaller companies the opportunity to increase the availability, performance and durability of its information.

The reasons mentioned above are just a few of the many that makes the internet one of the most powerful WAN and why it is essential that most server-client applications or services support efficiently this infrastructure.

S3FS provides an interface to interact with AWS and let the users of this file system implementation to interact with S3 repositories in a friendly manner. S3 repositories provide a secure and reliable storage service at levels that would represent huge costs for the companies with these needs. The advantages of the AWS relies on the on-demand services they provide, this way the users (companies) pay what they use. These kinds of situations make us realize the importance and significant value that solutions, able to support WANs, can contribute to the companies and other users.

1.2. S3FS

Although the current implementation of S3FS runs and works over WANs, it does not provide proper mechanism to deal with the problems inherent to this kind of networks. Between some of the problems not considered in the current implementation we have:

- *Performance*: Due to the latency and delays of the wide area network, most of the operations that require lots of readings from the file system will suffer a significant and unbearable performance impact.

- *Fail Recovery*: Errors, loss of packets, interruptions, are just a few of the many problems that can arise in the middle points of a wide area network. The current S3FS implementation could cause the entire mounted file system to hang because these problems are not being handled in a proper manner.

- *Multithreading*: Although the current implementation can handle several users working over the same file system, FUSE is not configured to work with multithreading capabilities. Therefore all the requests from all the users are handled in the same queue. If we would enable multithreading capabilities in FUSE the S3FS implementation would not be able to handle it properly.

1.3. Purpose of the project

In this project we will implement mechanisms to mitigate the first problem described, performance.

For this purpose we will implement an asynchronous caching mechanism that allows the clients to improve their experience with S3FS. Providing an asynchronous cache mechanism in the client side diminishes the impact of the wide area network latency. The user Application will notice a significant improvement in the performance as it will interact with the “local” cache instead of dealing directly with the S3 Amazon services.

The solution implemented will handle the request sequentially as the current S3FS version but asynchronously. The cache will provide and “instant” reply most of the times (Reads not present in the cache need to be retrieved from S3) and then it will perform the requests for its processing by S3. These requests could be added to a queue which will be served by a pool of threads interacting with S3.

Due to time constraints for the completion of the project the data structures to handle the caching system will not be fully optimized.

1.4. Limits

Due to time constraints, the goal of the project is to provide a significant improvement in the performance by providing some enhancements in the S3FS implementation. These improvements should increase the user application performance by covering the wide area network latency with a caching system.

The data structures implemented for most of the project might not be fully optimized for this purpose. Therefore is recommended to improve the resultant implementation if achieving the max performance increase is intended.

1.5. RELATED WORK

NFS[2] provides the capability to build a loopback file system, we will use an implemented interface based on NFS using FUSE[3]. This implementation will be modified to mitigate the wide area network latency effect in the performance of the current S3FS implementation.

Pangae[4] builds a unified file system over several servers spread around the world, therefore wide area network problems arise. The main goal of Pangae is pervasive replication, this means intensively replicate the data in nearby nodes of the system. Thanks to this exhaustive replication system and other improvement into the system, Pangae achieves fault tolerance; hide network latency and supports for disconnected operations. The idea of Pervasive replication is similar to persistent caching but with major improvements in the availability and reliability.

Wide area network are “slow” by nature due to several factors, the paper “A Low-bandwidth Network File System”[5] discusses a file system built to work under this circumstances. The basic principle of LBNFS relies on identifying similar blocks of data between different files or versions of the same file. Therefore avoids sending blocks of data that already exists in the remote server. The performance results show that LBNFS consumes one order of magnitude less bandwidth than other systems.

1.6. DESIGN AND IMPLEMENTATION

In the Figure 1 it is showed the high level design of the solution proposed, a Cache system will be implemented in order to reply faster to the client application. The cache will work with a queue managed by the LibAsync[] library. The LibAsync library provides us the infrastructure necessary to work based on events. This way we can attend other request meanwhile the request to S3 are processed (Writes are immediate replies because the write is done to the cache. Reads are not immediate replies if the data is not in the cache).

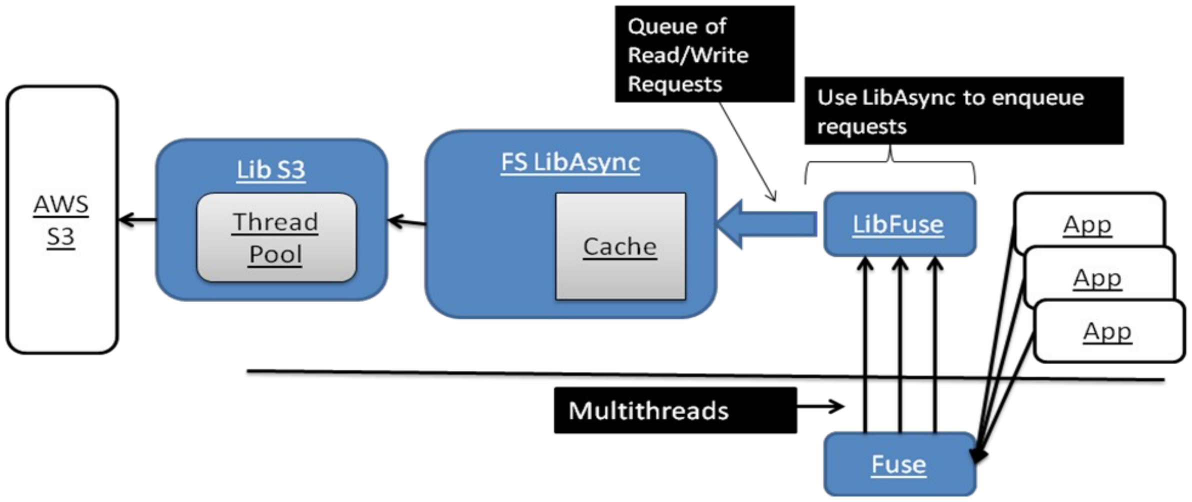

Figure 1. High Level architecture of S3FS with wide area improvements.

Once the Read/Writes are processed by the cache, it is queued so the threads in the pool are able to pick ne and process it using LibS3. Here is where the parallel processing happens; the different threads in the pool will pick one request to process it against S3.

1.7. Asynchronous messaging

Libasync was used to implement an asynchronous mechanism to communicate LibFuse and the S3 repository. Figure 2 clearly shows the interaction between the different modules and implementation of the system.

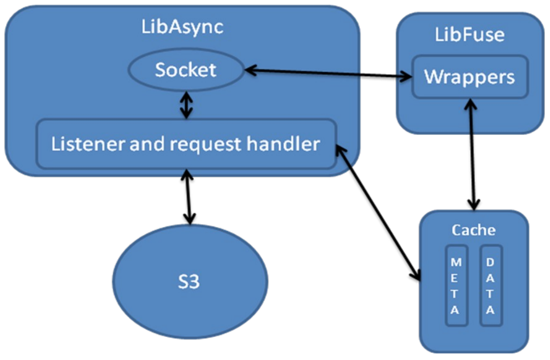

Figure 2. Asynchronous messaging

LibFuse used to handle the requests to S3, now there are wrapped functions which handle the request following the chart in Figure 3.

The requests are handled by wrapped functions in Libfuse, if the object is in the Cache, the operation is performed and the function returns the status of the operation. If the object is not in the Cache, then the request is sent to the Socket for further process. In this situation there are 2 possible behaviors, if it is a Read the client must wait for a response, if it is an update operation (Write, Remove), the entry in the Cache is updated, the request is sent, and the function returns a successful status. It is important to mention that if it is an update it is assumed that the object is in the Cache. A PinCount control should be implemented in the future as an improvement in this section. Only Cache entries with PinCount=0 should be removed from the Cache.

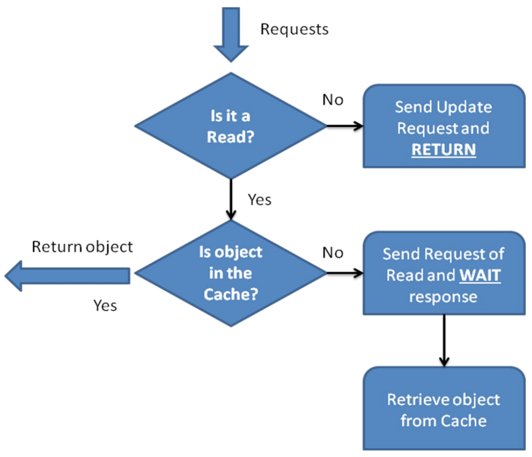
Requests
Is it a Read?
No
Send Update Request and RETURN
Yes
Return object
Is object in the Cache?
No
Send Request of Read and WAIT response
Yes
Retrieve object from Cache

Figure 3. Flow of the requests.

1.8. Cache

The caching mechanism is implemented in the first component after Fuse passes the requests. The system counts with 2 caches: meta-data, and data. Hash structures will be used to implement the cache for the three components. As S3 is a key—value based storage system, a hash structure using the S3 key is a perfect fit.

Each of the caches will have a fixed capacity therefore we need to implement a replacement mechanism to handle replacement when any of them are full. LRU replacement policy will be used which states that the most recently used element is most likely to be needed again in the future.

Therefore we will keep a linked list to control which elements were not recently used. An “old” element, i.e. has not been used for a “long time” will appear at the head of the list, whenever we need to make room for a new element in the cache we will choose a candidate to replace from the head of this linked list. Figure 4 illustrates the Cache interaction with the rest of the system

The elements of this linked list might be dirty because they received a write operation; therefore we need to have a dirty bit to control when to write the changes to S3 if the element is chosen for replacement.

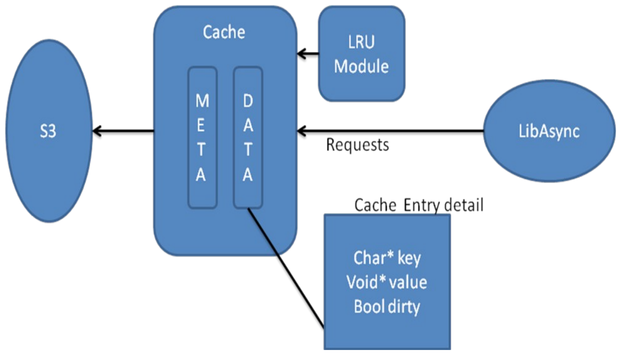

Figure 2. Detail of the Cache

The Read/Write request will be wrapped using the LibAsync library in order to handle the request asynchronously making use of events. Then they will be processed against the proper cache and then will be passed to the second component which will make the changes permanent against S3 repositories.

1.9. Thread pool

Although the thread pool was not implemented in during this project, it should be relatively easy to add this feature to the current implementation. The thread pool will allow us to handle parallel processing of the Read/Write requests passed by the first component. The number of threads will be configurable and set at the beginning of the program, the number of threads will be determined by the amount of work load of the first component. It is important to mention that the only Reads that are not already in the Cache will reach this stage.

Each thread will perform the same action, will pop an element from the queue created by the first component and will executed and finally send the reply to the first component. The writes will generate to replies but the Reads need to be passed to the cache component as fast as possible.

1.10. FUSE Multithreading support

Multithreading support will be enabled for FUSE in order to process the request with several threads and increase the performance improvement. Therefore we will need to verify that FUSE does handle all the necessary blocking mechanism to avoid executing actions in an incorrect sequence.

Without enabling multithreading in FUSE, it will not be possible to achieve the best performance results from this implementation. If FUSE handles all the requests with one thread,

EVALUATION

Due to some limitations of the system, stress scripts could not be run over the new implementation. But it was clear to notice the response of the system when executing read, write commands.

Although it depends on the connection latency and speed of the network, the average response for reading a 24KB metadata file with the original S3FS was about 2 seconds. With the current performance improvements the response is in the hundreds milliseconds or less. This last depends more on the speed and response of the local machine rather than the network latencies.

CONCLUSIONS

S3FS for the wide area it is a first attempt to show how many WAN's issues can be addressed in order to improve performance. The improvement achieved with the implementation of a Cache is similar to working with a local file system. In order to show the behavior with heavy workloads it would be necessary to implement the thread pool and run stress scripts against the system. Due to some limitations of the system the scripts were not run in order to gather information about the performance behavior; it was possible to verify the significant improvement in single and common tasks as Reads, writes, and removes of metadata information (when executing ls –l dirname).

S3FS is a basic implementation of improvements in order to mitigate the performance impact of the WAN in distributed file systems. The current implementation is a good example on how simple mechanism can provide significant performance improvements.

There is the need to introduce multithreading capability to the system and optimize the structures. Major tests are also necessary to prove the behavior of the system with heavy workloads.

REFERENCES

[1] S3FS implementation delivered in Lab 2, CS 6464, Cornell University.

[2] Sandberg Russel, Goldberg David, Kleiman Steve, Walsh Dan, Bob Lyon. Design and implementation of the Sun Network File System, Sun Microsystems. Mountain View, CA.

[3] File System in User Space, http://fuse.Sourceforge.net

[4] Yasushi Saito and Christos Karamanolis, HP Labs, Palo Alto CA.

[5] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. October 2001.