Can protocol and application layer statistics improve client-server responsiveness?

Jacob C. Thebault-Spieker
Computer Science
University of Minnesota, Morris
Morris, MN 56267
theba004@morris.umn.edu

Abstract

OpenAFS is a distributed filesystem, which allows users to have access to files stored within OpenAFS from any computer which has the OpenAFS client software installed. The client software ranks the servers that are a part of the OpenAFS network in order to decide which of the various distributed servers to interact with. The current ranking algorithm in the OpenAFS cache manager is either based entirely on the IP address of the servers, or has been modified to use network latency measurements that collected by the client-side network statics gathering mechanism. While using network latency as a metric for ranking the volume location server lists and file server lists that are maintained by the client software seems to help reflect current network conditions, there are other types of data that may be useful to consider. These include throughput, CPU load on the servers, and the number of clients interacting with a server at any given time. Additionally, this data could be distributed, providing a "current cell network conditions" service. This would allow clients to be able to rank their server lists using data collected from every other machine in the cell, creating a more accurate representation than what can be constructed by an individual client's Rx statistics. This paper describes ongoing research into the effects this type of service may have on allowing the client to rank on network quality. The goal is to devise a better ranking scheme, allowing organizations using OpenAFS in a WAN environment to have their server lists be modified to better reflect the current network conditions (e.g. a user on a mobile device, traveling).

Jacob C. Thebault-Spieker
Computer Science
Univserity of Minnesota, Morris
Morris, MN 56267
theba004@morris.umn.edu

1 Introduction

Traditional networked filesystems are a fairly simple client-server architecture, where all of the clients interact with one file server, or multiple individual file servers. However, when one of these file servers goes down, the files on it are inaccessible until the server becomes available once more. One way to deal with this issue would be to duplicate the files on multiple individual file servers, but this approach leads to an inability to ensure that all copies of the file remain the same. The ability for files to remain up to date becomes significantly more crucial if the files being hosted are critical to the organization.

This problem lead to the development of distributed filesystems. These filesystems distribute data across multiple file servers, and manage the replication of data automatically. This ensures that the copies of the files on all of the file servers remain up to date, which ensures that when a file is modified, all of the file servers are updated. Should a file server become unavailable (due to hardware failure, scheduled maintenance, etc.), access to the files stored on that server is not lost.

The capacity for replication of data leads to other interesting uses. For instance, if a multinational organization conducted business across country borders, prior to the advent of distributed filesystems, organizations would have had no ability to ensure that their data were available to offices on the other side of the world. Or, if they chose to use a networked filesystem, the system would be unusable because of network capacity. With a distributed filesystem, the organization would be able to locate some of their file servers in each geographic location where they may need access. This would mean that access to the files would be localized within the geographic location, but could be shared within the global scale organization.

Andrew Filesystem (AFS) was one such distributed filesystem. Originally a part of the Andrew Project (started in 1983) at Carnegie Mellon University, AFS was eventually bought by Transarc in 1989, which was later bought by IBM in 1998 and turned into IBM Pittsburgh Labs. In 2000, IBM released the source code to AFS, which allowed the beginning of the OpenAFS project. Because AFS is an old filesystem, there is some legacy functionality in the way OpenAFS works. One portion of this functionality is the way OpenAFS deals with deciding which server the client ought to interact with when finding a file the user has requested. The current way of making this decision is based on outdated network architecture assumptions, which leads, in many cases, to what amounts to a random decision about which server to interact with.

This paper will describe work-in-progress that attempts to solve this problem by distributing statistics about the overall network state, to provide the client with information about network quality and server state (packet round-trip time, throughput, server CPU load, server user load, etc.) in order to allow the client to optimize it's server connections. We are currently working to allow the client to use packet round-trip time and throughput (both data which can be collected locally), which provides a starting place for choosing servers based on network condition.

In Section 2, we attempt to describe the problem, discussing both background information pertaining to OpenAFS, and the current way that OpenAFS attempts to deal with the inherent issues in a distributed filesystem. We then propose an alternative solution in Section 3 by reconsidering the primary goals of the current way OpenAFS deals with optimizing the user experience in client-server interactions. In Section 4, we discuss the progress that has been made in implementing the proposed solution, and we describe a plan to finish implementation and begin evaluation of our proposed solution.

2 Current OpenAFS ranking mechanisms are based on outdated network architecture assumptions.

Within this section, we provide enough background information about AFS, and OpenAFS in particular, so that the reader is able to understand the problem. We first discuss important OpenAFS concepts and terminology, as well as how the OpenAFS distributed filesystem functions. We then discuss the pertinent part of the AFS protocol, Rx, which is an integral part of AFS, and is used for all communication between clients and servers. We then describe the problem with the operation of OpenAFS, namely that the client software often makes poor choices when connecting to a server, thereby leading to poor performance and poor user experience.

2.1 What is OpenAFS, and how does the ranking work now?

OpenAFS is an implementation of AFS (Andrew Filesystem), that distributes data within a cell.The cell is a logical concept, and does not affect the function of the network in any way. A cell consists of two types of machines, clients and servers.

Within the cell, there are many physical machines that run specific software to provide the functionality offered by a distributed filesystem. The OpenAFS software has a number of different types of server processes which are often run on different physical machines, including the file server process, the volume location server process, and the protection server process (manages the security aspects of AFS). There are other types of server processes that are needed to ensure that AFS will work correctly, but that is outside the scope of this document (these include a Kerberos process to handle authentication, a network time process to ensure all of the servers maintain the same system time, and others).

Traditionally, file servers host data on partitions, but AFS splits data further, into logical collections of files and directories called volumes. It is at the volume level that the distribution of the data takes place. Volumes are distributed across any of the file servers within the cell, and are not necessarily distributed to every server. Volumes are distributed on a read-only basis, and are most often mounted within a global namespace (within a Unix or Unix-like system, this is generally `/afs/cell_name/mount_point`. Within Win-

dows, the mount point is `\\afs\cell_name\mount_point`). OpenAFS does not distribute read-write volumes, and as such, there is only one read write point for each volume within the cell.

Once a user requests a file, the client software needs to determine which file server to contact to find the file that has been requested. However, in order for the client software to know which where the file is located, the client must know where the volume that contains the file is located. In order for the client to know which volumes are hosted on which file servers, it must request that information from the volume location servers (also called the VLDB, or the database servers). The database servers maintain a database of which servers are hosting which volumes. When a user requests a file, the client will poll the VLDB, which will give the client a list of file servers that are hosting the volume requested. The client must then pick a server from this list to request the file from. Each server is assigned a rank by the client, and this rank is used to choose the server from which to request the file.

Up until recently, server ranking assumed a specific network architecture, in which the IP addresses of the servers indicated something about the location of the machine to which the IP is assigned. If the IP address of the server was the same address as the client requesting the file, then the client was considered to be the same machine as the server, and the base rank given to the server was 5,000. If the IP address of the server was within the same subnet as the client, the rank given to the server was 20,000. If the server's IP address was on the same class of network (Class A, Class B, Class C), the rank given to the server was 30,000. If none of these conditions were met, the rank given to the server was 40,000. The base rank of 10,000 was reserved specifically for the database severs. Once the base rank had been assigned, the rank was modified slightly by adding a random number between 0 and 14, so that the number of servers with the same rank was minimized.

Currently this ranking occurs upon client start-up (either when a machine is booted, or the client service is restarted), and is never modified. Because it was assumed that IP addresses would not change, and that users would not be accessing AFS via a mobile device, the current ranking method has been forced out of date with the advent of laptops, cell phones, and the need for file access from a variety of locations. The current ranking mechanism never updates the server ranks until manually forced to. The key motivation for this project is to develop a new ranking scheme that is more flexible and dynamic, allowing for ranks to change as network conditions change and mobile users connect to the network in different ways.

2.2 Discussion of the problem

As has been stated, the way OpenAFS ranks the server lists is based on the assumption that the IP address of the server indicates something meaningful about where the server is located in relation to the client. While this assumption may still hold in same cases (e.g. an organization located in once place, using OpenAFS internally), because of the global nature of OpenAFS, and it's usage-cases in wide-area networks, it is possible for clients

and servers to be in a different geographic locations, and the IP addresses may be disjoint. If this is the case, the current ranking scheme within OpenAFS will rank all of the servers with a base rank of 40,000 (which is then slightly randomized to prevent excessive server rank duplication). What this amounts to is an essentially random ranking.

OpenAFS does provide a mechanism for the server administrators to manually define a rank, on a per server basis. However, this becomes unwieldy if an organization has a large number of servers within the cell, and there are significant inefficiencies with this approach. Additionally, this approach only addresses the issue if clients are not mobile, but the network is not built using the architecture assumed by OpenAFS. If clients are mobile, administrator ranked servers do not provide a better alternative to the automated ranking mechanism.

In order to attempt to solve this problem, one ought to consider the essence of the issue. That is, the fundamental issue with the previous way of ranking servers is that the assumptions made about how networks are designed no longer apply to current network architectures. Current network architectures, particularly wide-area scale networks, do not necessarily allow us to use IP addresses as a proxy for proximity between clients and servers.

Ultimately, it is not the location of the infrastructure, that is, file and database servers, that affects the quality of the experience of the user. Location seems to be a poor measure when attempting to optimize the user experience when a user interacts with remote servers. Therefore, if the goal is optimizing the user experience, one must consider the factors that affect user experience individually. These factors might include both factors about the quality of the network connection, and factors that relate to the ability of the server to provide a good experience. Upon defining these factors, one must consider how to better optimize these factors.

3 Proposed Solution

In this section we address the factors that affect user experience within OpenAFS. In discussing these factors individually, we both discuss the problem, and consider how these factors might be optimized, in order to provide the best possible user experience. We then consider the best way to solve the problem, and propose a solution. Subsequently, we describe the work that has already been done in order to provide a solution, and our intended next steps.

When a user requests a file, the request is sent across the network, and interpreted through the mechanism described earlier with this paper. Once the client decides which server to request the file from, the server must then receive the request for the file from the client, and serve the file to the client. The client then downloads the file, ending the transaction. There are three potential bottlenecks within this process, the initial transaction with the servers to request the file, the processing of the file request, and the actual serving of the

file. These bottlenecks ultimately fall into two categories: network bottlenecks and server performance bottlenecks. In order to optimize user experience we ought to minimize the occurrence of both types of bottlenecks.

3.1 The Rx protocol

The current implementation of OpenAFS uses Rx[3] as an underlying protocol, to provide basic client/server communication. Rx was chosen for a variety of reasons: it provides the abilities to make Remote Procedure Calls (RPCs), it ensures an authenticated connection between two machines (when an Rx connection is made), and it is extensible, allowing for the creation of custom security modules, and the definition of end-to-end encryption algorithms.

One of the side effects of using Rx is that a connection between client and server is viewed, on the network level, as a connection between two machines that are equal. That is to say, Rx makes no distinction between clients and servers, and both clients and servers are considered "peers" by Rx. Each peer is identified by an (IP address, UDP port) pair, and Rx maintains network information about a connection, and the peers on either side of the connection. The information maintained by Rx includes packet round trip time, packets transmitted and retransmitted, the amount of data sent and received, and other congestion information.

This information is currently used primarily by the `rxdebug` utility, to allow administrators to understand issues that may come up between client and servers, on a network level. Things like round trip time, the amount of transmissions compared to the amount of retransmissions, and congestion statistics may aid administrators in tracking down and resolving network problems.

3.2 What are the metrics that may affect user experience?

There are a number of different factors that may affect user experience when a client is interacting with with a server. We consider network latency, network throughput, server CPU load, and server memory usage. These are the factors that we believe believe are pertinent, but we don't consider these factors to be exhaustive.

3.2.1 Network Latency

When a packet is sent from a machine, UDP protocols require that the machine receiving the packet will respond with another packet. The amount of times this send-respond transaction takes is referred to as "round trip time" (RTT) by Jacobson[1]. Jacobson provides an algorithm for a round trip time estimation[1].

RTT provides insight into the amount of time it takes for the client to interact with the server, and for the server to respond. This is an important metric in user experience, as it indicates the "speed" of the network between client and server.

In order to optimize this metric, we would want to minimize the RTT between client and server. Therefore, allowing the clients to choose a file server with a low RTT ought to aid in providing an optimized user experience.

3.2.2 Network throughput

For the purposes of this paper, throughput is defined as:

$$\frac{\frac{numberofbytessent}{RTTofsendingbytes} + \frac{numberofbytesreceived}{RTTofreceivingbytes}}{2}$$

This metric can be used to indicate the transfer rate of the network connection, which in turn will affect the rate at which users can send or receive files to the file servers within the cell. Again, this is an important metric when trying to optimize user experience, given that the primary purpose of AFS is to allow users to store and retrieve files from the file servers within the cell.

Given the way throughput has been defined, a larger value for throughput indicates a better network connection, as it means a large amount of data was transferred over a short period of time. Therefore, choosing a server with a high throughput value ought to provide a better user experience than one with a lower throughput value.

3.2.3 Server CPU load

It is well known that if a computer is running with a high CPU load, other things attempting to run on that computer may be slowed down significantly, or may be required to wait until the CPU intensive process finishes. For instance, if a user were requesting a file from a server running a job that demanded attention from the CPU until it finished, and the job was running the CPU at full capacity, the user may not get their requested file in a timely fashion.

While it is unlikely that file servers are running CPU intensive jobs, it is possible that they may be running as web servers, or other servers that may see a lot of traffic. It is also possible that the hardware being used by organizations may be older hardware, so that something not considered CPU intensive on current hardware, may affect the ability of a machine using older hardware to manage the job.

3.2.4 Server memory usage

Similar to the CPU load metric, if a server is running a high memory load, it will begin swapping to the hard disk, which is significantly slower than accessing data stored in memory. It is possible that a server with high memory usage would suffer noticeable slow down.

Again, similar to the CPU load metric, it is relatively unlikely that a file server would reach a point where the memory usage is high enough to affect performance, but it is possible that due to hardware failure, or on a web server that sees a significant increase in traffic, a server may suffer a significant drop in available memory. This scenario may force the server to begin swapping to the hard disk, and negatively affect the user experience.

3.3 How to balance these metrics?

Some of these factors are more pertinent than others, but all may have an affect on overall user experience, as it pertains to connection quality. It seems clear that the more "network specific" factors will have a larger positive impact on the overall quality of the user experience. As of yet, it is unclear as to where the "sweet spot" is when deciding how much of an affect each factor ought to have, But it does seem that if a server reaches a point where the CPU usage or memory load are affecting user experience, the RTT, and possibly throughput, may be affected adversely as well. That said, a packet of data sent across the network is a much easier target to fulfill than sending a large file across the same network.

3.4 Providing a solution

In order to utilize these metrics, this paper proposes a distributed network quality service, which would allow clients and servers to share network statistics with all of the other machines in the cell. Currently, clients collect data via the Rx protocol, and expose this network data via the `rxdebug` utility. However, this data is only collected about servers that the client interacts with, which is too limited to make an informed choice about which servers ought to be used when data is requested.

With the goal of ensuring that all clients within the cell have as much information as possible with which to rank the server lists, a new distributed server process that maintains information on the current cell state, and notifies clients within the cell when requested would allow all clients to be able to make better decisions.

4 Progress made

While this work is still very much in-progress, significant progress has been made. Here we present both a description of the work that has been done, as well as a series of steps that we plan to take in order to complete this work. Additionally, we discuss potential issues

with the algorithm described, and propose a solution to these problems. We also propose a way of evaluating the impact of this work on user experience.

4.1 Planned steps

In order to implement this distributed server process, there are a number of steps that have been or will be taken:

1. Expose Rx level statistics to the local client software
2. Develop the algorithm to rank servers based on RTT and throughput, with an eye toward extending the algorithm to eventually include CPU and memory usage
3. Expose the server state statistics to the entire cell via distributed data structures
4. Expose the client side Rx statistics to the entire cell via distributed data structures
5. Modify the ranking algorithm to use data from the distributed data structures

4.2 Progress

This past summer, through Google's Summer of Code program, implementation began on exposing Rx level statistics to the client, as well as developing the algorithm to rank servers based on these statistics.

4.2.1 The distributed network analysis algorithm

Because of a design decision that was made within OpenAFS, any server with an RTT of 3,000ms or higher is considered to be not available on the network. Due to this assumption, our distributed network analysis algorithm also considers servers with RTTs of 3,000ms to be unavailable, and therefore ignores any RTT above 3,000ms. Servers with lower RTT values are given lower ranks, with the base rank being 5,000. The assigned rank increases from there on an exponential scale. That is, for each power of 2 that the RTT reaches, 5,000 is added to the rank. Thus, for an RTT of 2ms, the rank would be 10,000, for 4ms the rank would be 15,000, for an RTT of 128ms, the rank assigned would be 40,000, and for a maximum RTT of 3,000ms, the rank assigned would be 65,000).

When considering throughput, a similar scale (from 0 to 3,000) is being used, although the direct relation to availability of the server is no longer applicable. Work is being done to evaluate how to best treat throughput values, given that in many cases, the download capacity of the connection is much greater than the upload capacity, and clients ought not be penalized for their connection being the bottleneck on upload throughput capacity. For instance, a user interacting with an AFS cell from their home DSL connection (a common connection has 1.5Mbps download rates, and 256Kbps upload rates) may have a significantly different experience when downloading files from the remote server than their experience when attempting to upload files to the remote server. It is unfair, and potentially

detrimental to the user experience if users are penalized excessively because of their low upload rates.

4.3 Potential issues

A scale between 0 and 3,000 makes logical sense when the metric for server rank is exclusively RTT. That said, when the metrics used are expanded to include things like throughput, and particularly server load metrics, this relationship arguably no longer applies, and ought to be expanded to include other information. Because 3,000 is an arbitrary limit imposed by legacy code, it seems that this ought not to be the upper limit on the scale. Instead, the scale ought to be independent of the metrics themselves, and the metrics could be scaled to deal with any arbitrary limits that may affect them.

The current plan for dealing with this incomplete in relationship to the scale is to create an individual scale for each metric in a way that makes sense specifically to the domain. This will allow each metric to create it's own sub-rank for each server, and then these sub-ranks can be used to form a final rank for the server.

4.4 Proposed Evaluation

We have submitted an application to the Emulab Network Emulation Testbed[2], which will allow evaluation of the solution this paper proposes in an entirely emulated environment. We have requested between 4 and 8 emulated systems which will provide the ability to emulate wide-area network environment. The intent is to emulate an environment in which file and database servers are located across multiple geographic locations, with a variety of network conditions.

There will be two virtualized client machines; one will run a current production-ready version of the OpenAFS client, and one will run a modified version of the client software. These machines will be used to attempt to show a statistically significant difference between the previous way of ranking servers, and the proposed solution.

5 Conclusions and Future Work

Looking forward, this project aims to explore whether or not a distributed service that will report on the state of the AFS cell will aid in providing better user experiences. The approach presented within will provide an overall representation of the state of the OpenAFS cell, enabling client software to intelligently rank client-side server lists, which we believe will lead to better user experiences, particularly with mobile users that change their geographic location with some frequency. With the previous way of ranking server lists, the

client software run by mobile users may have assigned the worst base rank to the servers.

The project presented in this paper aims to resolve problems like this, and others, by dynamically re-ranking server lists, giving the clients an ability to modify the servers they interact with based on changes in network conditions, whether the quality of the connection changes due to a failure in network infrastructure, or whether the client, by nature of being mobile, modifies it's own network connection frequently.

5.1 Acknowledgements

The preliminary work that has been done for this project was funded through Google's Summer of Code 2009, and was done in conjunction with the OpenAFS community. In particular, large thanks go to Jeffrey Altman and Derrick Brashear for their help in both design and implementation of this work. As this work continues, OpenAFS will continue to be an integral resource. Further, thanks go to Nic McPhee, professor at the University of Minnesota, Morris for acting as the advisor throughout this process.

References

[1] JACOBSON, V. Congestion avoidance and control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols* (New York, NY, USA, 1988), ACM, pp. 314–329.

[2] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev. 36*, SI (2002), 255–270.

[3] ZELDOVICH, N. Rx protocol specification draft, 2002.