Vol.2011-HPC-129 No.7
2011/3/15

# StableSearch: A Searchable File Content Metadata System for the Gfarm File System

Joel Tucci† Osamu Tatebe†

Distributed file systems are allowing scientists, corporations, and users to access and share data on a scale previously unimaginable. The utility of these systems cannot be understated, but there are still many problems to be solved, not the least of which is helping users quickly search the petabytes of stored data for the files they are interested in. This problem has largely been solved on the desktop, for example Apple's Spotlight search, but these approaches are not scalable to thousands of nodes. Obviously there is a need for a search tool that can quickly search these files without negatively impacting the running system or giving a user access to metadata for data they don't have access to. To this end, we have designed StableSearch, a tool for searching Gfarm[1] distributed file systems. We have shown that our tool is capable of locating data significantly faster than grep while having a minimal impact on the performance of the running system.

## 1. Introduction *

We are in the midst of a data explosion, petabyte scale systems are now common and now researchers have their sites set on Exabyte-scale systems[2]. Distributed file systems such as Gfarm accomplish this by federating anywhere from a few to thousands of individual storage nodes into single logical file system. Data is also replicated across nodes in order to both increase the reliability of the file system as well as to increase performance by taking advantage of access locality.

While these tools allow us to store data on scales previously unimaginable, our abilities to find content users find interesting in these systems is not keeping up. Many users are still using tools like grep in order to find the proverbial needle in a haystack. Brute force searches such as these not only take a lot of time, they put additional strain on the metadata and file servers as they must read massive amounts of data that the user isn't actually interested in before finding something the user will find interesting.

In order for scientists to better be able to manage data, find data they are interested in, discard data that they are not, and avoid duplication of effort, we need to develop a better way to search these distributed databases. Current efforts are either not scalable or not designed to run on the same file system that the data is processed on. In order to solve these problems we are proposing a distributed file system search tool we are calling StableSearch. Stablesearch is designed to allow users to quickly sort through massive amounts of data stored on a large number of servers without negatively impacting the running file system.

## 2. Related Work

### 2.1 Desktop Search

Content-based metadata searches are already commonplace on the desktop. Tools such as Apple's Spotlight[3] and WinFS[4] essentially create an on-disk database that analyzes file content then creates a map of search terms to the files that contain those files. In order to keep the database as up to date as possible they use kernel-level libraries that alert them whenever a file has been updated on disk. The search agent daemon then analyzes the file, and if the file isn't on it's ignored list it parses the file's contents and then updates the on-disk database accordingly. Since the agent gets alerted to every change on the disk there is no reason

* † University of Tsukuba Department of Computer Science

to constantly comb the file system looking for updates. And since the databases are able to throw out very common and repeated words, they remain quite compact and can be efficiently searched.
Furthermore these system impose the same security standards as their host operating system. Even though users would never be able to open files they do not have access to, knowing whether or not a file even contains certain terms can be damaging to the owner of said file. Therefore these systems are very strict on only letting users see results for files they are allowed to view, although due to many of these system's proprietary nature it's hard to discern what the exact security mechanisms are.
While these systems work well for desktop searches the approach used isn't very scalable. Namely they require that a daemon be alerted every time a file is changed and that the same daemon parse every file. On a distributed file system with many nodes the daemon will constantly be requesting files that are stored on another node, quickly saturating the network and becoming a bottleneck. Furthermore because it has to request so many files the daemon may not be able to keep up with file updates in a timely fashion, a crucial feature for search.

### 2.2 Scientific Metadata Searches

Recently a lot of research has been done on creating repositories of searchable scientific metadata, such as the Scientific Annotation Middleware(SAM) system[5] developed by the US National Center for Supercomputing Applications. Systems such as these are invaluable for cross-domain sharing of data and saving time and money by avoiding carrying out redundant experiments. This system uses a WedDAV interface to share data amongst disparate institutions. The advantages of using the web are simple, everyone has access and the data can be searched and stored by multiple search engines allowing a simple, consistent interface to the data at a low cost to the institutions housing the data. Users can use search engines to scour data sets before downloading anything, thus saving bandwidth and time on both sides.
However this systems main strength can also be its main weakness, namely that since the data is designed to be shared it isn't necessarily designed to be stored where the experiments are being carried out. There are many occasions where data may not have to be shared with the outside world, such as intermediate results and experiments, and furthermore the closer your data is stored to where you are using it, the faster your system becomes. So therefore it would be incredibly advantageous to have a searchable metadata system running on the same systems that are storing and processing the data.

### 2.3 Magellan

Magellan[6] created by Leung et. al is an interesting new way to quickly search file metadata, such as file size, owner, name etc. It is a replacement for the Ceph[7] distributed file system's metadata server and uses files instead of databases for metadata management and search. Magellan is able to quickly locate files with the given characteristics without a big footprint and moreover is able to quickly throw out results that almost certainly will not be interesting. However Magellan does not currently keep track of file content metadata, which can be quite useful when trying to comb through terabytes of data.

## 3. Design and Impleentation of StableSearch

### 3.1 Design Goals

The overall design goals of StableSearch are as follows:

- To be able to enable/disable the search tool as needed.
  - As a corollary to that, keep changes to the Gfarm code to an absolute minimum.
- To minimize impact on overall file system performance.
- To make searches as accurate as possible.

Since the content search will not be necessary in every Gfarm system the ability to turn it off is crucial, although the impact isn't large it isn't zero and for extremely large scale scientific experiments turning it off is a necessity. Along those lines, a system that did not modify the Gfarm code would be the most desirable since it would allow the two systems to evolve somewhat independently. Obviously we want to limit the performance impact as well. Accuracy is probably the most difficult of the stated design goals. Ultimately any content metadata search is going to be probabilistic, it is only possible to index so much metadata before the databases supporting the metadata gets out of hand. Furthermore in systems that are geographically dispersed it may only be possible to query a subset of all the servers. However thanks to file replication it is usually only necessary to query a subset of all the file system nodes in order to search every file.

### 3.2 System Architecture

The overall architecture of StableSearch is outline in Figure 1:

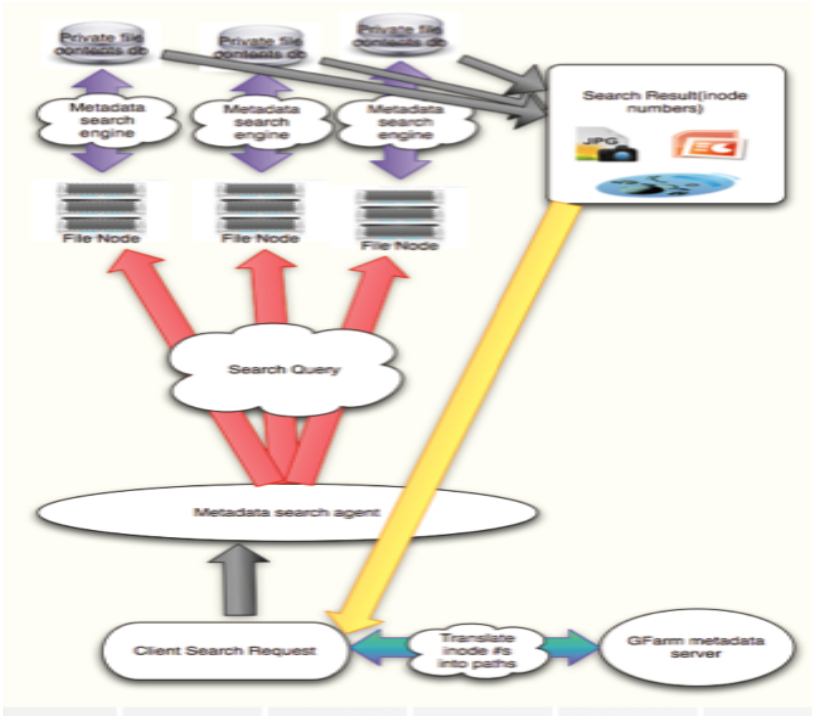


**Figure 1 Outline of StableSearch**

### 3.2.1 Design of the Metadata Store

We first start off with the observation that locality is critical to the performance of distributed file systems, as the network can quickly become a bottleneck, especially for storage nodes connected over disparate locations[8]. Reading from the local disk is almost always faster than reading the same data across the network and thus minimizing network traffic is essential to good performance. In order to essentially eliminate any extra network overhead incurred by indexing data we decided to leave managing the content metadata databases up to the individual file nodes. Each file node has a copy of a metadata search tool installed on it and configured to listen for updates to files stored in the Gfarm data directory. Whenever a file is modified on a storage node, the update daemon running on that node updates its private metadata database, thus eliminating any need to use the network to communicate metadata updates.

### 3.2.2 Design of the Search Agent

The next critical piece is the search agent, the process responsible for taking input from the users, querying the file system nodes, and returning results. The overall process can be split up into three main steps:

1. Query the file system nodes
2. Translate results into logical file names
3. Ensure data integrity.

For the first step the agent takes input from the user then broadcasts out the query to all the file system nodes. The nodes then read the broadcast and perform a metadata search on their local database. They then send back a list of inode numbers corresponding to all the files that match the requested query. The search agent then waits for either a certain number of responses or a certain time period to elapse then collects all the results, removes any duplicates that may have occurred due to replication of the data, and then uses the Gfarm metadata server's database to translate the inode numbers into logical path names. In order to do this in a reasonable amount of time we must first make sure that the Gfarm metadata database has reverse lookups for the inode number allowing us to see the file name and parent directory. From the parent directory when can then recursively search backwards until we get to the root node.

Now we could just return this list to the user, however we have a problem, we don't necessarily know if the requestor has read access to the files involved. While the metadata results would not allow the user to see the file contents, it would allow them to determine what files contain what terms, which can be good enough to do damage in some cases. So the search agent, which has the root key to the Gfarm metadata server's database, performs some additional permissions checks to ensure that the user only sees files they have read access to.

### 3.2.2 Implementation

We decided to use the Tracker[10] desktop search tool for Linux as our metadata backend for several reasons. First the system is incredibly extensible, not only is it open source but it also has pluggable metadata search libraries. Therefore it is ideal for eScience applications where we have a large variety of data formats.

For the search agent and file system daemons we used a Java messaging based approach using the HornetQ[11] JMS implementation. The agents run on each of the file system nodes and wait for requests. When they get a request, they query the Tracker database and then return

the inode results as described above. The search agent then uses the Java LDAP libraries to turn the inode numbers into readable file names.

## 4. Evaluation

All evaluations were carried out on a cluster of 11 machines(5 file nodes, 1 metadata server, 4 clients and 1 search node), the specifications of which can be seen in Figure 2:

| | |
|---|---|
| CPU: | 2 2.4 GHz dual core Xeon processors |
| Memory: | 6 Gb |
| Hard Disk | 1 500 Gb 10000 RPM drive |
| Operating System | Cent OS 5.5 |

**Figure 2 Evaluation Hardware**

### 4.1 Gfarm metadata performance

The first thing we evaluated was the effect of the extra index on the metadata server, the most important server in the Gfarm file system. The results are summarized in Figure 3. The average creation time for a directory went from 14.5 microseconds to about 16 microseconds, an increase of around 10%. This is within tolerable range for certain groups of applications.

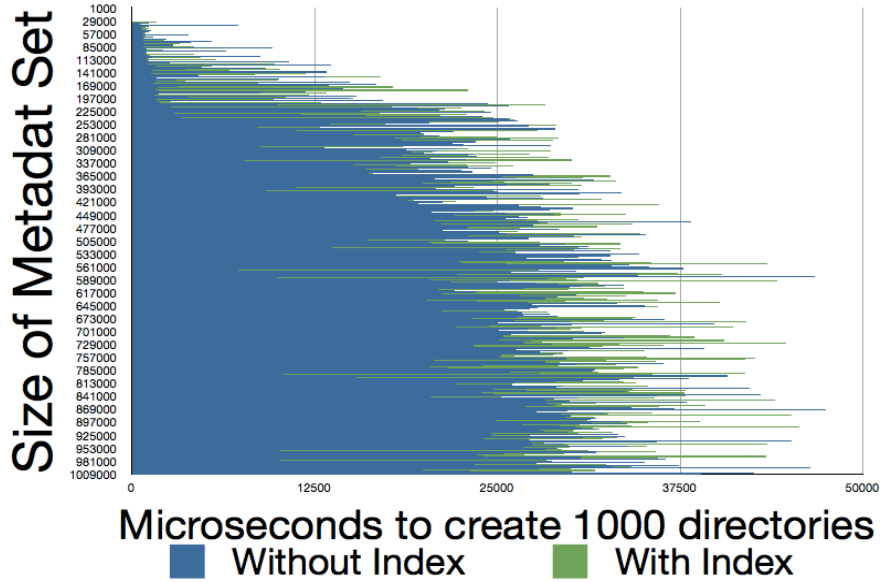


**Figure 3 Evaluation of adding the entryINumber metadata database index**

### 4.2 File System Performance

In order to assess the performance impact, and efficacy of the search query we ran experiments copying the entire English Project Gutenberg[10] DVD from April 10, 2010 to the metadata server 4 times, with each of the client nodes copying the entire DVD once. We did it with three different configurations:

- No search enabled, this is the baseline.
- Search enabled on one node using the FUSE[12] kernel module to mount the Gfarm file system, this tests how well a current desktop search engine would do indexing a Gfarm file system.
- Search agents enabled on each of the file nodes, this is our proposed setup.

We chose to use Project Gutenberg because it contains a large number of freely available material that is written using standard English and is something that people may be interested in searching.

Figure 4 shows the results of our experiments.

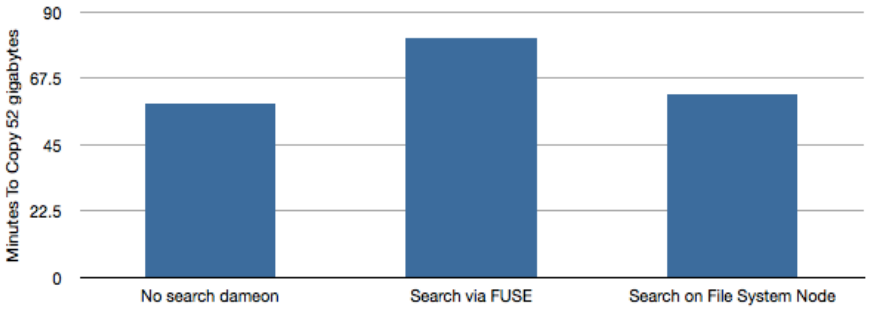


**Figure 4: Time to copy the files with the various search**

We can see that there is a very small impact from adding the search daemon to the file system nodes, only about 3%. Furthermore, when we use tracker to try to monitor a

remote mount, we do not receive notifications every time a file is changed. Therefore that approach is especially hard on the metadata server as it has to constantly query for lists of files. This is obviously not ideal and thus our experiments confirm what we have thought, a centralized metadata database alone is not practical for large-scale distributed file systems.

In this particular test the performance is mainly metadata server bound as the files tend to be small but numerous. While there are larger binary files, Tracker seems to have very little effect on their performance as it is designed to ignore binary blobs that it doesn't understand. What metadata that is extracted out of files such as video or audio files, such as artist information or frame rates, can usually be found

### 4.3 Search Performance

Now that we have seen what overhead the search adds, lets actually evaluate the results of the search itself. As stated earlier, the search is essentially a probablistic one. Tracker extracts a certain number of "interesting" terms and stores it in the metadata database. Increasing this amount will increase the potential number of results you get, but increases the cost to store, update, and search the database as well as increasing the size. For this experiment we stuck with the default values for the database size as we are trying to get a feel for the potential of the system. For this experiment we searched for the word "whale" and measured the time it took and the number of results given.

Figures 5 and 6 show the results of our tests in terms of the number of results returned and the time it took to return them. Grep obviously returns "perfect" results, but those results come at quite a cost in terms of time. While books are of general interest, there are plenty of other files that could be interesting to users and scientists alike. The number of hits here could easily be increased by plugging in an appropriate ontology, something that is supported by the Tracker software.

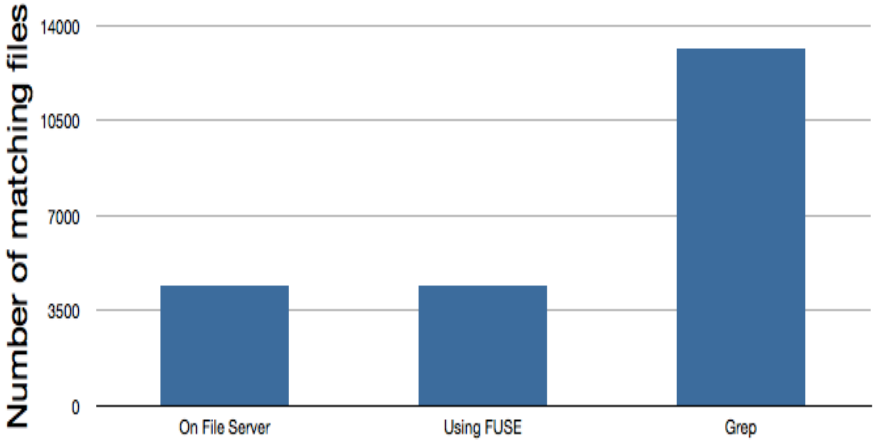


**Figure 5 Number of matches for the word "whale"**

Figure 6 shows the time it takes to do the various searches on a logarithmic scale. As we expected the overall times are not even close, grep took about 3688 seconds to complete, the search when we had Tracker simply scan the Gfarm file system via FUSE took only about 1 second since the database was totally stored on the local disk. In comparison, StableSearch took about 16 seconds to go out and query the file systems and convert the inode numbers back into logical file paths.

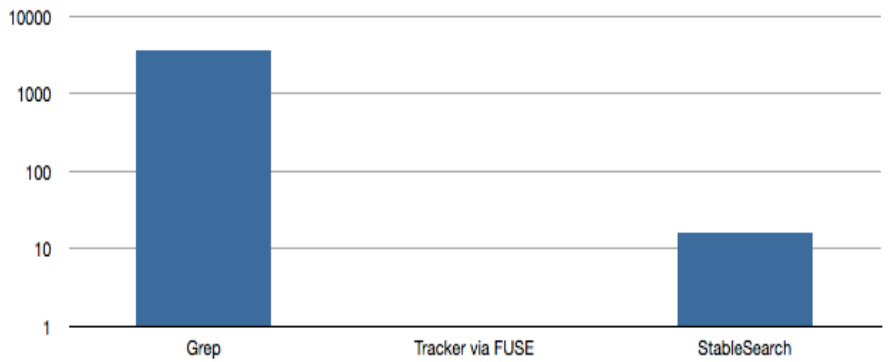


**Figure 6 Comparison of search times for different search methods.**

## 5. Conclusions and future work

We have shown that it is in fact possible to create a searchable distributed file system with minimal impact on the performance of the actual file system while keeping queries quick and relatively low cost. This is achieved by bypassing the network for most queries and instead keeping the content database local to the file servers.

However there is still room for improvement, namely on search accuracy and metadata performance. While a 10% degredation in performance may be acceptable for file systems mainly designed for personal computing use, it is not as acceptable in an e-science environment where performance is critical. In addition to increasing metadata performance in the future we would like to develop and evaluate domain-specific ontologies and evaluate how effective they are in helping scientists pinpoint data.

## 6. References

1) Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. Gfarm grid file system. *New Generation Computing*, 28:257–275, 2010.
2) Aloisio, Giovanni and Fiore, Sandro. Towards Exascale Distributed Data Management *International Journal of High Performance Compututing and Applied Computing*, Vol. 23, No. 4, pp.¥ 398--400
3) Apple Developer Connection. Working with Spotlight. http://developer.apple.com/macosx/tiger/spotlight.html, 2004.
4) T. Rizzo and S. Grimaldi. Data access and storage developer center: An introduction to “WinFS” OPath, 2004.
5) Schuchardt, Karen L. and Gibson, Tara and Stephan, Eric and Chin, George. Applying Content Management to Automated Provenance Capture, *Concurrency and Computation: Practice and Experience*, Vol.~20, No.~5, pp.¥ 541--554 (2008)
6) A. Leung, I. Adams, and E. L. Miller. Magellan: A searchable metadata architecture for large-scale file systems. *Technical Report for the University of California, Santa Cruz*, UCSC-SSRC-09-07, 2009.
7) Weil, S. A., Brandt, S. A., Miller, E. L., Long, D.D.E, Andmaltzahn, C. Ceph: A scalable, high-performance distributed file system. *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)* (Seattle, WA, Nov. 2006), USENIX.
8) Brahm Peter, The Coda Distributed File System, *Linux Journal,* Vol.~1998, No.~50es
9) Gnome Project. MetaTracker http://projects.gnome.org/tracker/
10) Project Gutenberg. http://www.gutenberg.org/wiki/Main_Page
11) HornetQ Proejct. http://www.jboss.org/hornetq
12) The Fuse Project. http://fuse.sourceforge.net/