# TOWARDS SCALABLE SEARCHING OF DISTRIBUTED FILE SYSTEMS

Itua Ijagbone, Shivakumar Vinayagam, David Pisanski, Kevin Brandstatter, Dongfang Zhao, Ioan Raicu
iijagbon@hawk.iit.edu, svinayag@hawk.iit.edu, dpisan2@uic.edu, kbrandst@hawk.iit.edu, dzhau8@hawk.iit.edu, iraicu@cs.iit.edu

## Abstract

Scientific applications and other High Performance applications generate large amounts of data. It's said that unstructured data comprises more than 90% of the world's information [IDC2011], and it's growing 60% annually [Grantz2008]. The large amounts of data generated from computation leads to data been dispersed over the file system. Problems begin to exist when we need to locate these files for later use. For small amount of files this might not be an issue but as the number of files begin to grow as well as the increase in size , it becomes difficult locating these files on the file system using ordinary methods like GNU Grep [8], which is commonly used in High Performance Computing and Many-Task Computing environments. We tackle this problem of finding files in a distributed system environment by using our model. Our work leverages the FusionFS [1] distributed file system and the Apache Lucene [10] centralized indexing engine as a fundamental building block. We designed and implemented a distributed search interface within the FusionFS file system that makes both indexing and searching the index across a distributed system simple. We have evaluated our system up to 64 nodes, compared it with Grep, Hadoop, and Cloudera, and have shown that FusionFS's indexing capabilities have lower overheads and faster response times.

## FusionFS

- FusionFS [1] is a distributed file system that co-exist with current parallel file systems in High-End Computing, optimized for both a subset of HPC and Many-Task Computing workloads.
- Distributed metadata management is implemented using ZHT [2], a zero-hop distributed hash table.
- ZHT has been tuned for the specific requirements of high-end computing (e.g. trustworthy/reliable hardware, fast networks, non-existent "churn", low latencies, and scientific computing data-access patterns).
- The data is partitioned and spread out over many nodes based on the data access patterns. Data is indexed, by including descriptive, provenance, and system metadata on each file.
- FusionFS supports a variety of data-access semantics, from POSIX-like interfaces for generality, to relaxed semantics for increased scalability.

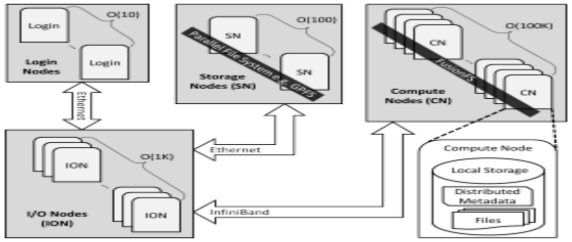


Figure 1. FusionFS deployment ina typical HPC system

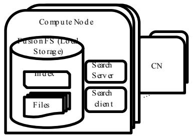
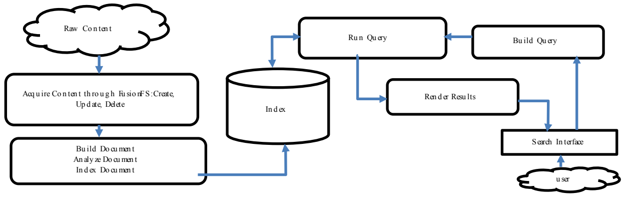


Figure 5. Index and Search interfacedeployment on FusionFSin atypical

## Lucene

- Lucene [10] is a high performance, scalable Information Retrieval (IR) library developed by Apache.
- Lucene provides search capabilities to an application. It's a mature, free, open-source project implemented in Java.
- Lucene provides a powerful core API that requires minimal understanding of full-text indexing and searching.
- In Lucene, Documents are atomic unit of indexing and searching. It can index and make searchable any data that in which text can be extracted from.

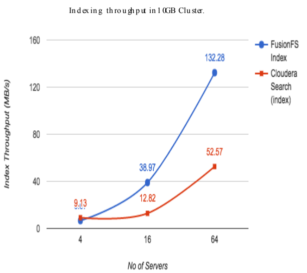


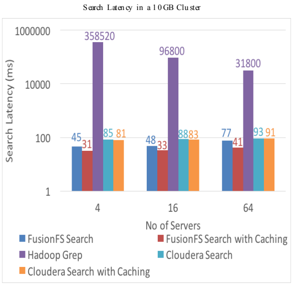
Typical Components of a SearchApplication as seenthrough FusionFS

## Testbed

Our index and search interface was written in the C/C++ programming language. CLucene is in C/C++ as well as FusionFS. We had two different machine environments for testing and evaluation:

- **Local Virtual Machine:** We have explained why we did this in Section 4.2. Our testbed for this environment was a 64-bit Virtual Machine with 2 vCPU and 1.5GB of RAM.
- **Amazon Elastic Compute:** For deployment to a cluster and comparing it to other similar implementations as explained in Section 4.3, we used Amazon's Elastic Compute Units. We deployed our FusionFS implementation, Grep and Hadoop Grep on a High Frequency Intel Xeon E5-2670 v2 (Ivy Bridge) Processors with 2 vCPU, 7.5GB of RAM and 32GB SSD storage for the 10GB data cluster. For validation against Cloudera Search, Cloudera Search ran on the same Intel Xeon with 8 vCPU, 30GB of RAM and 160GB SSD storage.
- **Metrics and Workloads.** Our testing metrics covered the following: Writing Throughput, Index Throughput, Search Throughput and Search Latency. We explain each of these metrics in Section 4.2. Our workloads were divided into two sections.
- **Local Workloads**: This is workload we ran on the local virtual machine. Our testing data was generated from an English Dictionary. The total size of the data was 1GB but was split into 100MB files (10 files in total). In other words, our workload for this experiment was based on weak scaling, keeping the node constant and increasing file sizes from 100MB to 1000MB (1GB). Experiments for the search latency and search throughput were repeated at least three times. The reported numbers are the average of all runs.
- **Cluster Workloads**: This workload was run on the Amazon Cluster. Our testing data is a 10GB Wikipedia dataset [25]. The 10GB dataset are in chunks of 64MB, this chunks are distributed evenly among our 10GB cluster. This workload was applied to our implementation, Grep, Hadoop Grep and Cloudera Search. For search latency and throughput, we ran a 1000 queries where we found the average.





## Index Throughput

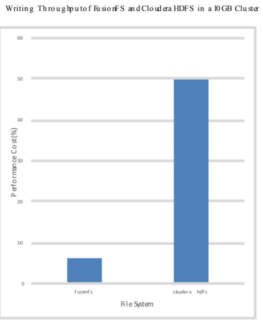
- The index throughput is the speed (MB/sec) at which we can index the files in FusionFS as the servers increases.
- We see that as we increase the number of nodes, FusionFS does much better than Cloudera Search by at least 2.5x.
- This is because as we scale the number of nodes, the workload on each node is reduced.
- This however doesn't appear to be the case with Cloudera Search, increasing the number of nodes has little impact on the indexing throughput.

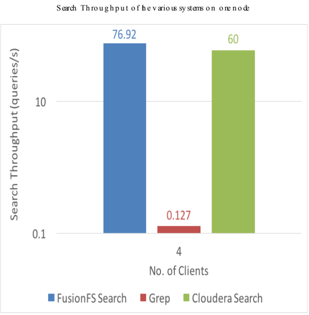
## Search Latency

- We compared Hadoop Grep, FusionFS Search and Cloudera Search on a 4, 16 and 64 10GB Cluster.
- Figure 14, shows Hadoop Grep performs the worse of all the implementations because Hadoop Grep counts how many times a matching string occurs and then sorts the matching strings.
- Our search latency when not cached and when cached does better than Cloudera Search when not cached and when cached respectively. One reason could be because we are running vanilla Lucene.
- Cloudera Search runs Apache Solr, a child of Lucene, which we believe has extra syntactic sugar added on top of Lucene. We also evaluated the search latency on FusionFS Search on a single node as we exponentially increased the data size. We see from Figure 15 that as the data size increases exponentially, the search latency grows linearly.
- We see that as we increase the number of nodes, FusionFS does much better than Cloudera Search by at least 2.5x. This is because as we scale the number of nodes, the workload on each node is reduced. This however doesn't appear to be the case with Cloudera Search, increasing the number of nodes has little impact on the indexing throughput.

## Write Throughput

- Finally, we wanted to know if adding indexing to FusionFS caused a drop in performance and if it did by how much.
- We compared the writing throughput of FusionFS when the indexing feature is enabled and vanilla FusionFS (without the indexing feature). We also wanted to know if there was a drop in performance with respect to writing to Cloudera HDFS as indexing was on going.
- This experiment was conducted on 10GB cluster made-up of 4 nodes. Figure 17 shows that our index feature reduces the throughput of FusionFS by an average of 6% while writing to Cloudera HDFS as indexing was happening showed a performance dropped of 50%.
- This drop in write performance in Cloudera can be attributed to the centralized metadata management of HDFS.





## Conclusion and Future Work

- We introduced a new system that extends the capabilities of FusionFS. Our system indexes and provides an interface for searching data stored in the cluster in a fast manner.
- We gave an overview of how search engines work especially Apache Lucene and how we used it in our work. We explained how we index files in FusionFS and how we make searching the file system possible.
- We showed with evaluations and experiments that our implementation is scalable, provides high throughput and has reduced latency. We also compared our implementation to Hadoop Grep and Cloudera Search, and showed through results that our system is better.
- We are also looking into the possibility of adding SQL API so that researchers who are familiar with SQL can run SQL like queries on our system.
- Another planned work is building a REST API. Since our target are researchers in the Scientific Community, we want to make our search process more user friendly by producing a simple to use web browser interface like Google Search