XFS - File System Dynamics Simulator

Arkaditya Verma(averma16@hawk.iit.edu) Siddhanta Biswas (sbiswas10@hawk.iit.edu)

***Abstract*—Computer clusters have evolved throughout the years, expanding into larger supercomputers and vast computer grids beyond our time. Due to their immense computational capabilities and versatility, computer clusters have become the preferred platform for scientific research. Scientists and engineers have invested a great deal of time and resources into improving these High-Performance Computing (HPC) systems, driving along the way the advanced development of computer technology and scientific discovery. With such improved HPC systems, there is a need for building a distributed indexing application that would help engineers and scientists to perform search and extract relevant information from large scale storage systems.**

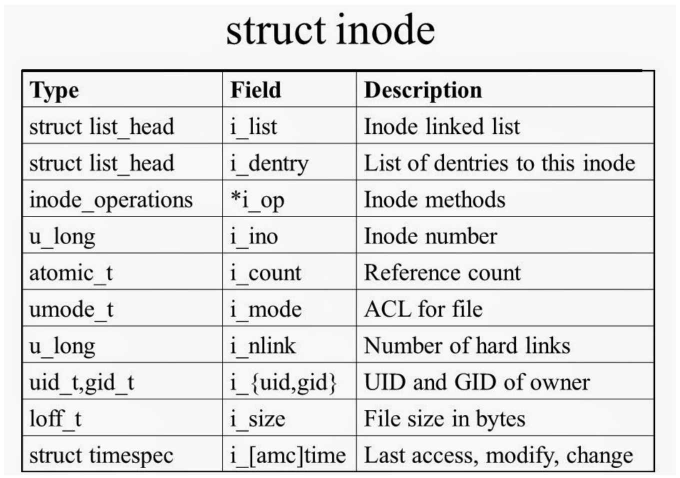
Keywords—simulator,throughput,filesystem,metadata,inode, XFS, Userid, framework, chunksize

Introduction

Metadata provides information about a certain item's content. For example, an image may include metadata that describes how large the picture is, the color depth, the image resolution, when the image was created, and other data. A text document's metadata may contain information about no of lines in the document, the author details, creation date, and a short summary of the document.The principal purpose of metadata is to help users find relevant information and discover resources. In operating system, a filesystem is the methods and data structures that the OS uses to keep track of files on a disk or partition. It describes the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Here , we have introduced design and prototype of a FileSystem Simulator framework (XFS Simulator) aimed at parsing file system dumps from various supercomputers and simulates the updates in the system at different granularities such as hours, minutes, seconds. In future , the simulator should be capable to decide if the indexing can be performed, given the dynamics of the file system and the availability of resources.

Background

Information retrieval is the process of finding relevant materials of an unstructured nature that satisfies an information need from within large collections of information resources. The recent developments and research in information retrieval area is mostly based on web page-based retrieval and search engines for such retrievals and local node search engines for faster and optimized file search. There is no such baseline information retrieval system which focuses on filesystem metadata search with storage capabilities and which supports multi-node tiered architecture. Considering all these issues and the bottlenecks which are not addressed by the current information retrieval systems, we aim towards designing a file system simulator which can be fit for the scientific research and other purposes. In a file system data is stored inside of files. The set of all management information necessary for the filesystem to operate is the *metadata*. Management information internal to the filesystem is the internal metadata while external metadata is made available to higher layers by means of an interface (eg. attributes or directory structures). Each file in the UNIX filesystem has an inode containing metadata about that file.The *inode table* consists of inode for each file and directory in the filesystem.


struct inode

Type	Field	Description
struct list_head	i_list	Inode linked list
struct list_head	i_dentry	List of dentries to this inode
inode_operations	*i_op	Inode methods
u_long	i_ino	Inode number
atomic_t	i_count	Reference count
umode_t	i_mode	ACL for file
u_long	i_nlink	Number of hard links
uid_t,gid_t	i_{uid,gid}	UID and GID of owner
loff_t	i_size	File size in bytes
struct timespec	i_[amc]time	Last access, modify, change

Fig: Data structure of inode in linux kernel

As part of our work, we are limiting our analysis to just throughput of the simulator, no. of files modified and accessed, no. of files based of ownership using the userid and indexing time for each data of the metadata dump.

Proposed Solution

As part of this project, we have designed and implemented a basic prototype simulator which provided with metadata file system dumps as input would analyze and simulate the modifications done as part of those file systems at granularities of modified time in hours and days, access time for the file or directory and user ownership and the no of files owned. The simulator developed till now is based on a single thread . It takes the data dumps chunks as command line arguments which when passed with necessary arg option would generate the output by extracting specific metadata fields from the filesystem dump and plotting the graphs based on the data points. The file chunk size analyzed is in the range of 1 gigabyte to 50 gigabytes. The application is developed using C/C++11 for parsing the data dumps as part of the simulation and Python 3 with matplotlib and numpy libraries to clean up the data files and represent real time graphs. Firstly, the data is cleaned removing unnecessary garbage characters from the single 200GB metadata chunk. The cleaned file is later split into chunks of 1GB, 10GB and 50GB.The program is run and performance analysis on these set of chunks. As its single threaded program, the throughput remains almost similar i.e as the size of chunk is increased , the time to read and index file increases. We tried to scale our application by implementing multithreading using queue where a single producer thread enqueues and multiple consumer threads dequeues from the shared queue using standard thread library (std::thread). But it suffered performance degradation , worst than sequential reader and writer. This is mainly due to poor performance by the std::queue data structure ,which was a bad idea for implementation. We are yet to try the multithreading performance using POSIX threads. As of now, the python and C/C++ modules are not tightly integrated and needs to be ran manually using scripts. This part of integrating the complete models has been left as part of future work for the project.

Evaluation

The experiment has been performed on two different set of chameleon cluster nodes: 1. Baremetal *Storage node* with 2 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz CPUs, RAID configured 15 2TB storage devices,64 GiB memory and 2. Baremetal *skylake* node with 2 Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz CPUs , 192 GiB memory and 240GB SATA ssd. As mentioned it is a single threaded application, the performance of the program is best when size of chunk is smaller so 1Gb and 10GB data chunk show better performance compared to 50GB and 200Gb data. We have found that on average filename makes the 65% of the metadata size.

Indexing Time:

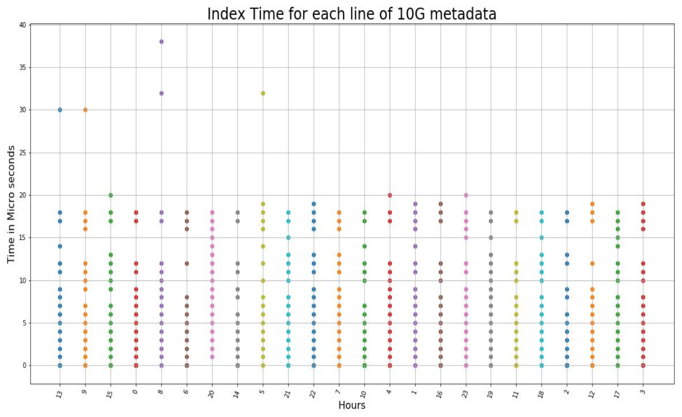


Fig: Indexing time for each metadata based on access time for 10GB chunk

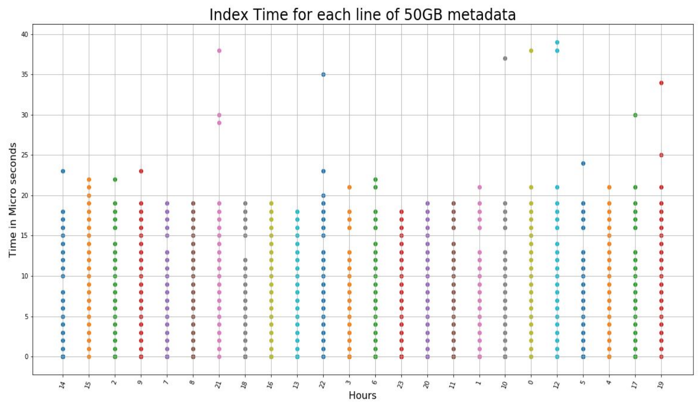


Fig: Indexing time for each metadata based on access time for 50GB chunk

The above diagram explains that as there are more files accessed during specific hour irrespective of the year or month the indexing time increases proportionally. Here HashMap data structure is used to store key value pairs.

Throughput:

The Throughput remains the same for every program run. As the size of the files increases, the time taken to read and indexing also increase proportionally due to single thread of execution. This is where we are working on the improvement which would also lead to faster indexing and run time performance.

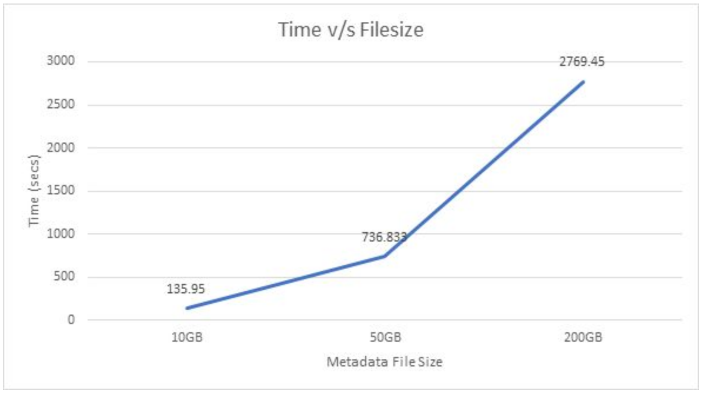


Fig: File read and index time for metadata chunks

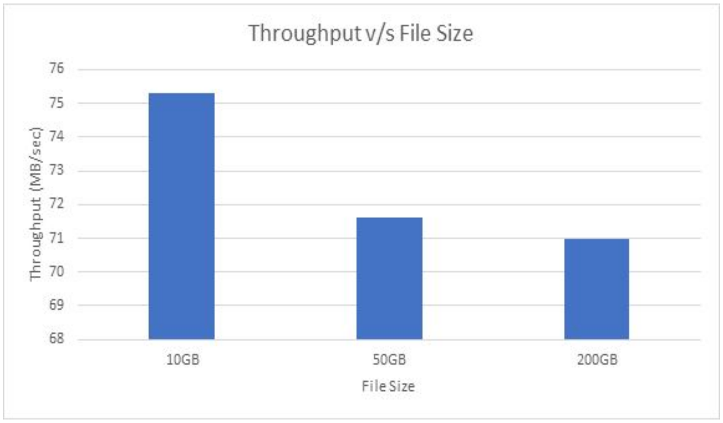


Fig: File read and index throughput for metadata chunks

Files Accessed and Modified:

The epoch system time was converted to human readable time (DD-MMM-YYYY HH:MM:SS) format into a new metadata chunk which was later passed to the simulator program. In Linux, *access time* represents the last time file was read. When passed as argument, the program gets the total no of files accessed using the access time field of the chunk. It can be seen that both the below graphs have very similar plots as they are part of the same metadata chunk and files accessed during 20 hours of the day for each day in span of 5 years remain the maximum.

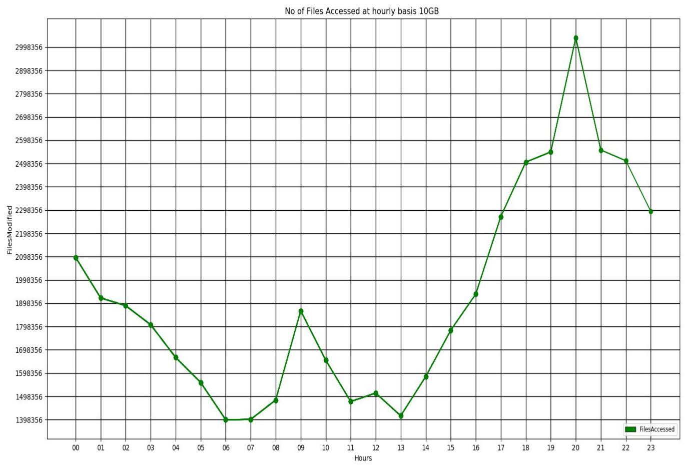


Fig: No of files accessed during specific hour for 10GB chunk

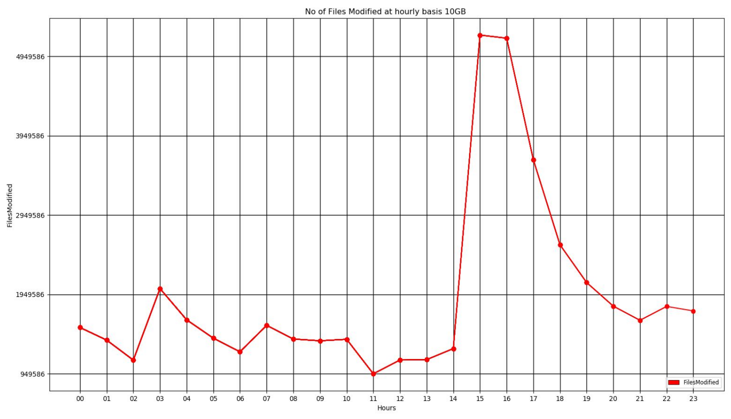


Fig: No of files modified during specific hour for 10GB chunk

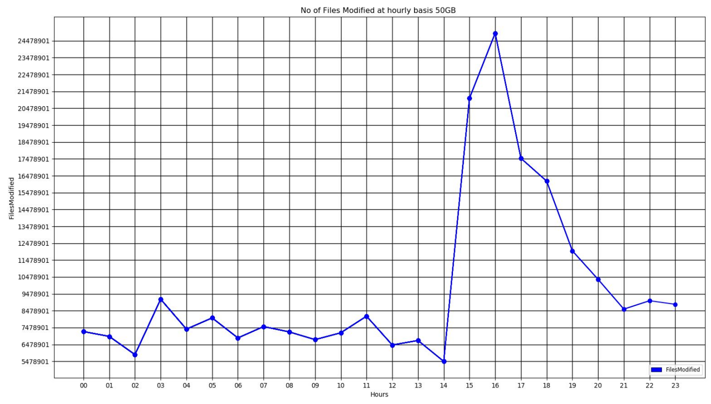


Fig: No of files modified during specific hour for 50GB chunk

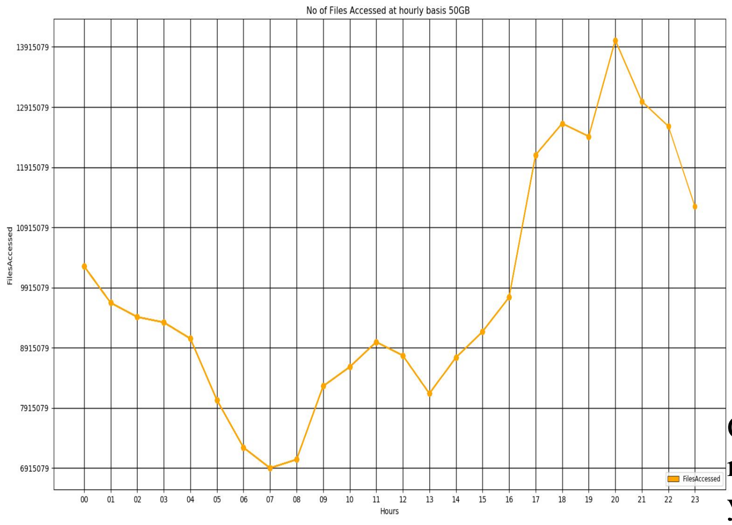


Fig: No of files accessed during specific hour for 50GB chunk

Modification time, represents the last time the file content was modified. It can be seen that both the graphs have similar plots here which represents that mostly on 16th hour of any day more no of files were modified on the supercomputing system and through which we can predict that usage of the system might be very high during this time of the day.

Other than the above, we have also analyzed the total no of files accessed on each day during a span of 10 years. This is a very dense graph as it consists of more no of files and dates in compressed format.

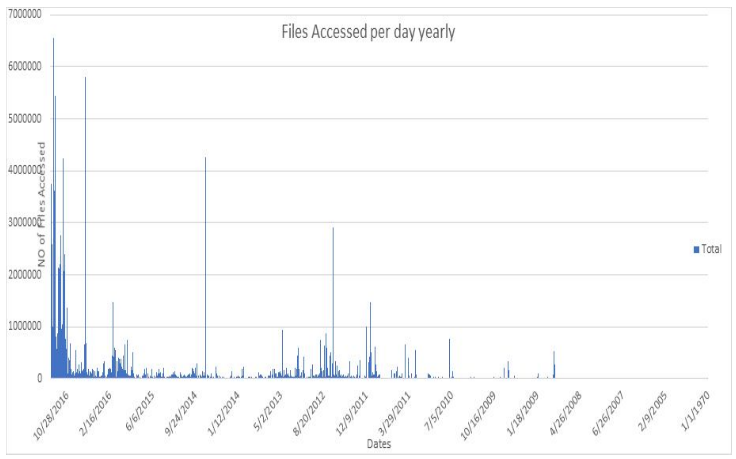


Fig: No of files accessed during specific day within 50GB chunk

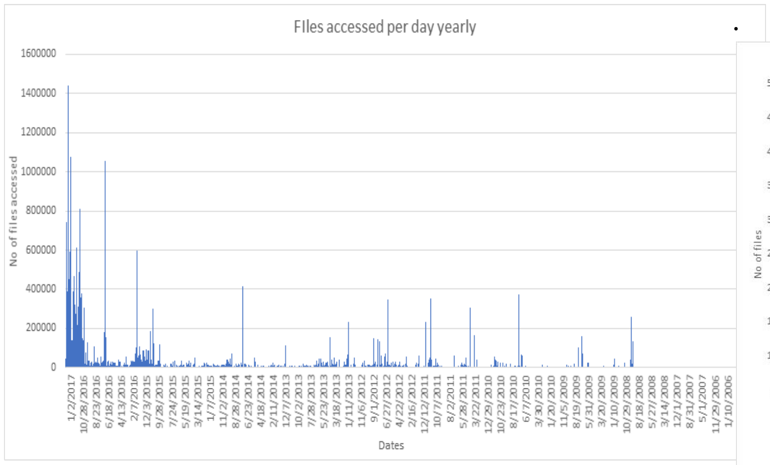


Fig: No of files accessed during specific day within 10 GB chunk

The above graphs show that during the time span of 2015-2017 most no of files were accessed as part of storage device on the supercomputing system.

User Ownership:

A *user ID* (UID) is a unique positive integer assigned by a UNIX operating system to each user. Each user is identified to the system by its UID, and user names are generally used only as an interface for humans. UIDs are stored, along with their corresponding user names and other user-specific information, in the */etc/passwd* file. Analysing the metadata dumps based on unique UID, gave us more better perspective of the ownership of files in the filesystem. It can be seen from the below graphs that approx 70 % of the total no of files are owned by the root user (UID 0), followed by user with UID in range of 58000. The root user is the one who has administrative privileges on the system and hence very obvious of the file ownership

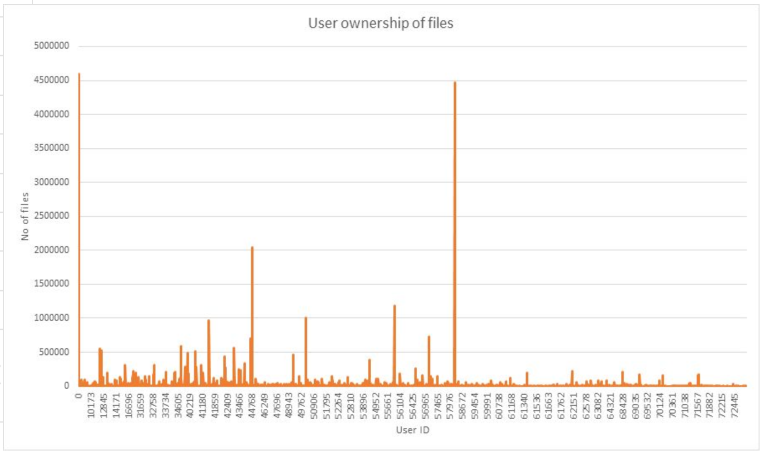


Fig: No of files accessed during specific day within 10GB chunk

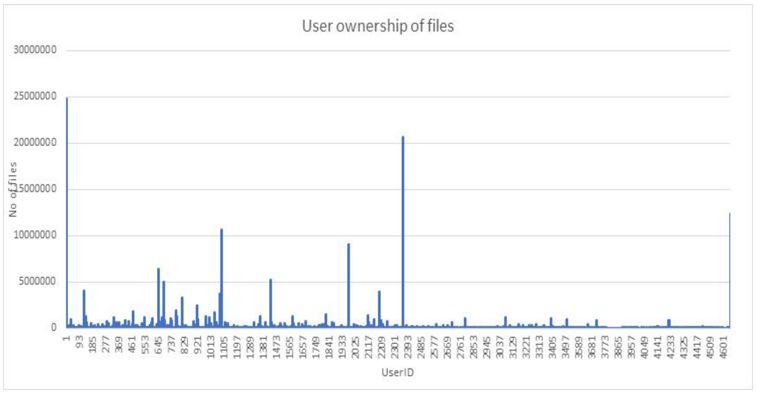


Fig: No of files accessed during specific day within 50GB chunk

From the above plots ,we can evaluate more information about the time when the system is mostly used by the users, what could be the performance of the disk based on how many files were modified and accessed within the filesystem at specific point of time. Some other plots which would have been more helpful include file size v/s UID plot which gives us more details regarding the storage usage of each user in the filesystem.

Design Challenges :

The major challenges faced during the framework design include:

1. Limitation of storage space on the chameleon cluster nodes, due to which we could only

keep upto 300 GB of data including the original metadata dump.

2. The performance of C++ standard thread library and standard queue on which we implemented multithreading and it showed worst degraded performance limiting the application to work on single thread instead .

Related Work

There have been multiple works done as part of File System simulator in the field of parallel and distributed file systems. Some of the examples include MOSS File system simulator and IMPIOUS (Imprecisely Modeling Parallel I/O is Usually Successful). IMPIOUS enables file system designers and researchers to try out innovative data placement strategies and other novel subsystems at scale,facilitate file system deployment by providing a low-cost platform for workload and file system tuning scenarios, empower scientists to quickly tune existing file systems for specific workloads, and aid instructors by providing a platform for in-classroom experiments. A perfect Simulation framework plays important role in evaluation of file system design, disk behavior and workload performance in computer systems. The File system simulator design can be based out of three categories: content based,metadata based and trace-driven. These simulation models help system researchers and designers to simulate large system on relatively smaller systems for experimental purpose and tuning of resources and help in tuning existing file system for specific workloads. Another open source tech which helps in managing large file systems is Robinhood Policy Engine. It replicates a filesystem metadata in a MySQL database which further allows querying of metadata within the database instead of the file system.

Future Work

As part of the future work,the simulation framework would focus on the indexing of the file system metadata to provide us with indexing throughput and given the availability of resources, if it is feasible for the indexing simulator to index the file system provided with such large chunks of data dumps. Also, working on scaling the application for better performance and so that it can keep up as the data sets increase in size. One of the major work would be to integrate the C/C++ and python modules, so that it supports automation of metadata dumps processing ,analyzing and generating real time graphs.

Conclusion

The Framework designed till now is a small prototype which gives more analysis based on file count ,user ownership and file access on daily and yearly basis. The data structure for indexing on metadata needs to be explored using existing and custom built data structures other than the hashmaps which are used as part of this project. If Scaling is supported , it would enable us to explore ways how can we make this feasible by adding more resources and making it distributed.

References

[1] Itua Ijagbone, “Scalable indexing and searching on distributed file systems”. Master thesis. Illinois Institute of Technology, 2016

[2] Alexandru Iulian Orhean, Itua Ijagbone, Dongfang Zhao, Kyle Chard, Ioan Raicu. “Toward Scalable Indexing and Search on Distributed and Unstructured Data”, IEEE Big Data Congress 2017

[3] Alexandru Iulian Orhean, Kyle Chard, Ioan Raicu. “XSearch: Distributed Information Retrieval in Large-Scale Storage Systems”. Oral qualifying exam. Illinois Institute of Technology, 2018

[4] E Molina-Estolano, C Maltzahn, J Bent and S A Brandt, “Building a parallel file system simulator”, Journal of Physics: Conference Series 180, 2009

[5] Chandramohan A Thekkath, Joh Wilkes, Edward D Lazowska, “Techniques for File System Simulation”, https://doi.org/10.1002/spe.4380241102, 1994

[6] Ray Ontko, “MOSS File System Simulator - Modern Operating Systems, 2nd Edition”

[7] “Robinhood Policy Engine”, http://wiki.lustre.org/Robinhood_Policy_Engine