Parallel Graph Reduce Algorithm for Scalable File System Structure Determination

Matthew Andrews*, Jason Hick*, John Emmons†, Kirby Powell‡
*National Energy Research Scientific Computing Center
†Drake University ‡St. Edwards University

***Abstract*—In recent years, high performance computing (HPC) has become integral to continued advancement in almost all scientific disciplines. Consequently, computing centers have increased their data processing and storage capabilities. However, back up algorithms employed by these facilities to prevent data loss have not kept pace. This is exacerbated by the increased number of error-prone storage devices composing modern file systems; the higher failure rates of these large systems requires more frequent back ups, limiting productivity and continued scientific growth. In this project, a computationally efficient and scalable algorithm was developed to speed up a necessary component of the back up procedure: file system structure determination. By comparison, many widely used algorithms determine file system structures with linear time complexity achieving good performance on small file systems; however, most cannot scale to multiple compute nodes, thus are limited by system clock speeds and the latency of random file system access. This poster describes the theory, benchmarking results, and implementation of this project's scalable, graph reducing algorithm with Apaches MapReduce framework, Hadoop. With the MapReduce framework, this algorithm can be massively parallelized using many of compute nodes, achieving a logarithmic time complexity when sufficiently scaled. Also, unlike other file system structure determination algorithms, this project's graph reducing method uses previously determined file system structures in current runs, reducing the number of file system access requests, thereby providing researchers with the maximum potential of an HPC facility's resources.**

Background

Despite the use of efficient back up algorithms, the large file systems of today's HPC facilities can still take several hours to days to back up. Researchers have focused their attention to a known bottleneck of the back up procedure, the determination of the file system structure. Determining the path to all files and directories, the file system structure, is key since to retrieve a file or directory's data the path must be known. In order to address this bottleneck, the host of this project, National Energy Research Scientific Computing Center (NERSC), originally implemented a parallel, tree-walk algorithm that determines the path with linear time complexity.

Fig. 1 illustrates the time complexity of the file structure determination algorithm used at NERSC, called tree-walk.

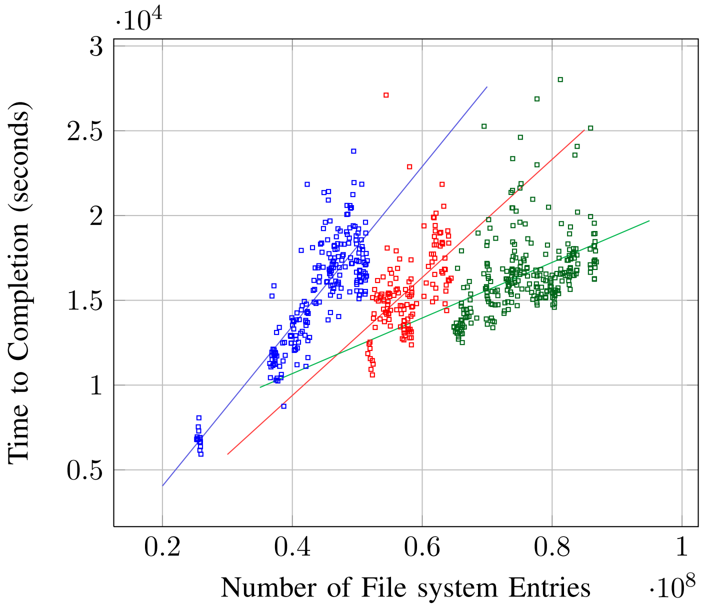

Fig. 1: Tree-walk acting on a file system with upgrades.

The strong linear correspondence between time to completion and file system entries implies the time to determine the structure of a file system is directly proportional the size of the filesytem.

Note the three curves have different slopes; this results from the algorithm's strong dependence on the speed of random access to the file system. Since the speed of the parallel file system at NERSC is directly dependent to the number of storage devices, the tree-walk algorithm runs faster when the number of the storage devices in the file system is increased. The leftmost curve shows the performance of the tree-walk algorithm on the initial implementation of the Project U2 file system. The center and rightmost curves show the performance of the algorithm as the number of storage devices were increased during upgrades to the same system. As expected, the slopes decreased as the file system access time descreased.

Despite obtaining a linear time complexity, the tree-walk algorithm contains major inefficiencies inherit to its operation: it is limited by file system access time, and cannot scale beyond a single compute node.

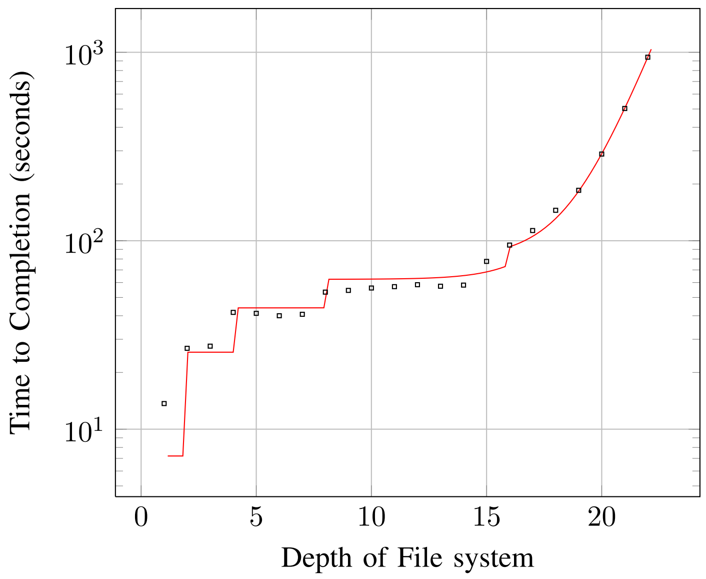

Fig. 2: Time to completion for analysis of a uniform file system tree with uniform branching factor two.

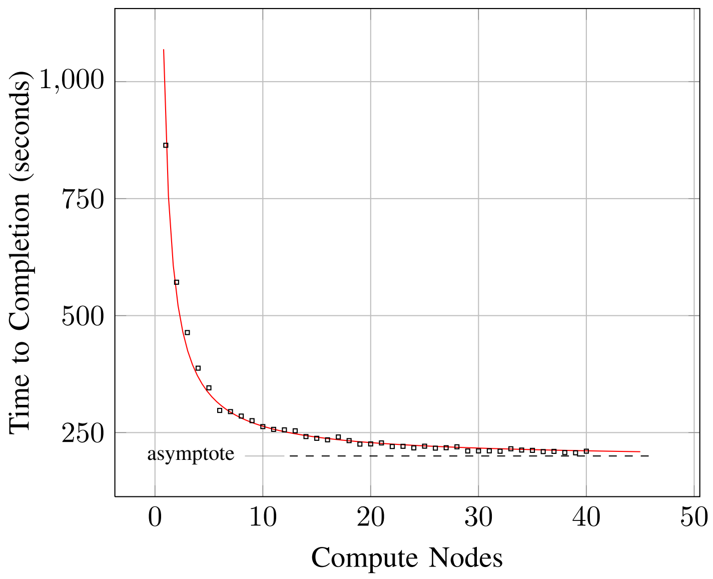

Fig. 3: The time to completion data for analysis of a constant sized filesystem.

Algorithm Design

The improved algorithm described in this poster is implementated with Apaches MapReduce framework, Hadoop. With the MapReduce framework, this algorithm can be massively parallelized using hundreds of compute nodes, achieving a logarithmic time complexity when sufficiently scaled.

The logarithmic time complexity is the result of the graph reduce method used in the algorithm. Rather than determining a structure by starting at the root of the file system and tranversing the entire tree, like tree-walk, this algorithm expands the structure outward from each entry in the file system. In this way, the structure is built by iterating the graph reduce algorithm $\lfloor \log_2(\text{depth}) \rfloor$ times, where the depth is the maximum depth of the file system tree.

Finally, Hadoop allows the graph reduce algorithm to incorporate additional compute nodes to speed up the rate of file system structure determination. This is not possible with the tree-walk algorithm which is a simple applicaiton that can only run on a single compute node. As a result, even though there is additional set up time with the graph reduce algorithm, it is still faster since the work can be spread across many processors.

Algorithm Performance

Fig. 2 illustrates the graph reduce benchmarking data taken on an artificial file system; this file system has a tree structure with a uniform branching factor of two, meaning each node split into two more nodes at each level. When isolated, the first part of the data has a logarithmic step relationship between file system depth and time to completion, as expected in the algorithm design; however, this relationship breaks down as the depth is increased further. This can be explained by the exponential relationship between the number of file system entries and file system the depth. To reiterate, if the same data was used, but plotted the number of file system entries v. time to completion, the logrithmic growth would still be present for small numbers of file system entries, but would quickly converge to a linear relationship as the number of entries increased.

After basic benchmarking data was taken, the number of compute nodes used to the process data was increased. Fig. 3 shows how the time to completion decreased as the number of nodes was increased. The file system that was analyzed was of constant size (depth twenty three and uniform branching factor two) containing enough file system entries such that the setup and cleanup time of the algorithm was neligible.

The inverse relationship between time to completion and the number of compute nodes is consistent with the time complexity data provided by fig. 2. Since the file system size was constant, the data was partitioned into smaller and smaller pieces as more compute nodes are added to the job. Interestly, as the number of compute nodes become arbitrarily large, the time to completion approaches a nonzero asymptote. This results from the inescapable setup, cleanup, file system reads, and intermediate data processing required as part of Hadoop and the graph reduce algorithm

Conclusion

In conclusion, the graph reduce algorithm has many advantages over the tree-walk algorithm. It capitilaizes on historically under-utilized resoures, yields significant performance improvements, and can be scaled much more easily and with less human intervention.