Parallel Graph Reduce Algorithm for Scalable File System Structure Determination

John Emmons, Kirby Powell, and Matthew Andrews, Jason Hick

Department of Mathematics and Computer Science, Drake University, Des Moines, IA 50311, USA

Department of Computer Science, St. Edward's 3001 Congress Ave, Austin, TX 78704, USA

National Energy Research Scientic Computing Center, 415 20th Street Oakland, CA 94612, USA

Abstract

High performance computing is essential to continued advancement in almost all disciplines of science. Computing facilities that process scientific data have increased their data storage and processing capabilities to meet the growing demand; however, preventing data loss across large arrays of error-prone storage devices limits growth when naïve approaches are employed for filesystem back ups. This article describes a scalable, parallel algorithm implemented to determine the structure of filesystems, a necessary component of the back up procedure, which uses Apache's Hadoop MapReduce framework.

Introduction and Motivation

FIG. 1: Tree-walk algorithm acting on NERSC Project U1 file system with three upgrades; each upgrade increases the number of drives and decreases file system access time.

- Parallel file systems (e.g. GPFS) strip data across hundreds to thousands of hard drives to decrease the read and write times of data. (more drives $\implies$ faster reads/writes)
  - Files and directories are not stored using a tree structure; rather, an inode (associated metadata) is used to track the location of the file across the drives.
  - Presents challenges to backup algorithms that rely on the path to retrieve data.
- National Energy Research Scientific Computing Center (NERSC) uses a multi-threaded applications, Tree-walk, to generate paths to all files before each backup.
  - Tree-walk is linear time-complex, but **does not scale** to multiple compute nodes! To increase backup speed using Tree-walk, the file system must be made faster.
  - File system backups where being significantly slowed due to limitations of Tree-walk.
- This project sought to develop a file system structure (path) determining algorithm that could utilize multiple compute nodes.

Algorithm

```
Data: curr_filesystem_data
for inode in curr_filesystem_data do
    if inode is a directory then
        inode_flag←false, sort_key←inode_parent
        write inode
        inode_flag←true
        inode_parent←sort_key←inode_self
        write inode
    else if inode is a file then
        inode_flag←false, sort_key←inode_parent
        write inode
end
```

```
Data: partitioned_mapper_output, filesystem_mount
children_parent←null
for inode in partitioned_mapper_output do
    if inode_flag is true then
        children_parent←inode_parent
    else
        inode_parent←children_parent
        write inode
        if inode_flag is true then
            report iterate_algorithm←true
end
```

- Algorithm uses a MapReduce paradigm (Apache Hadoop). Hadoop jobs involve two major steps:
  1. Data is split up among available compute nodes; each node run a user-defined script, the mapper, on its data.
  2. Data is then recollected, sorted/partitioned, and redistributed where nodes apply a second script, the reducer, on their data.
- Builds paths to files and directories **iteratively**.
  - Expands paths for all file system entries simultaneously in a step-by-step manner.
  - Child files and directories utilize parent path information to expedite path construction.
- Reuses old file system backups to retrieve paths to unchanged files and directories, dramatically reducing file system IO.

Performance

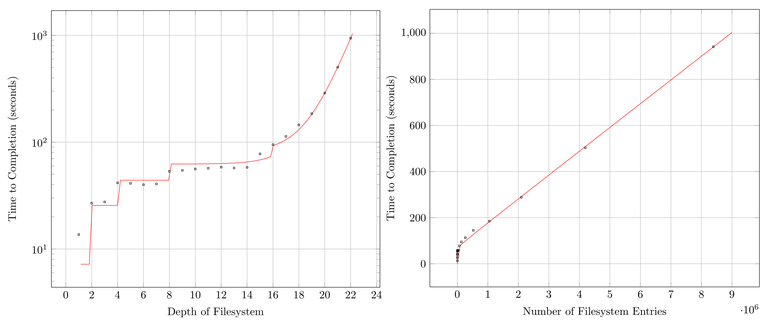

FIG. 2: LEFT: Time to completion for analysis of a uniform filesystem tree with uniform branching factor two plotted with *depth*. RIGHT: Time to completion for analysis of a uniform filesystem with branching factor two plotted with *entry counts*.

- **Logarithmic time complexity** when sufficiently scaled. Time to completion depends on the maximum depth of the file system (i.e. most nested directory or file).
- Degenerates to linear time complexity in the worst case (same as Tree-walk).

Optimization and Scalability

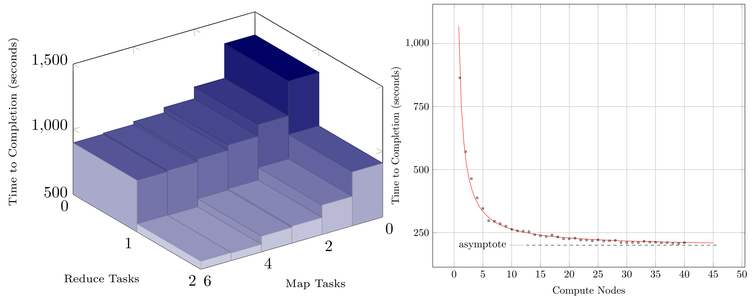

FIG. 3: LEFT: Analysis of a constant size filesystem with varying map and reduce task on a single compute node. RIGHT: Time to completion for the graph reduce algorithm analyzing a uniform file system tree with varying number of computer nodes.

- Optimal number of mappers and reducers per compute node is 2 and 6 respectively.
  - This result was expected since compute nodes have 8 processors and Hadoop allows a maximum of 2 mappers and 6 reducers per node.
- Tests of scaling were done on an artificial file system.
  - File system was uniform (branching factor 2 and depth 23) to ensure each node had the same amount of work.
  - The algorithm **scaled almost perfectly** as the number compute nodes increased.
- Scaling broke down as many more nodes were added, likely a result of increased communication and set up time.

Discussion

- This graph reducing algorithm is superior to Tree-walk is two significant ways:
  1. It can scale to multiple compute nodes; a much cheaper way to scale than with additional disk.
  2. It has Logarithmic time complexity when sufficiently scaled.
- Can be improved by implementing with Hama, a native parallel graph reduce paradigm, rather than with Hadoop.

References

[1] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters.," in *FAST*, vol. 2, p. 19, 2002.

[2] "Apache hadoop." `http://hadoop.apache.org`. Accessed July 2013.

[3] S. Seo, E. J. Yoon, J. Kim, S. Jin, J.-S. Kim, and S. Maeng, "Hama: An efficient matrix computation with the mapreduce framework," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 721–726, IEEE, 2010.