On-demand Indexing for Large Scientific Data

Brian A. Madden Aleatha Parker-Wood Darrell D. E. Long
Storage Systems Research Center, University of California, Santa Cruz
{*madden, aleatha, darrell*}*@cs.ucsc.edu*

Introduction

Enabling search alleviates the need for manual file management, allowing users to find files by the features that are most relevant to them. Despite the need for a comprehensive file system search, most file system indexing work has focused on adapting existing solutions such as the RDBMS or spatial trees to index files. Although a step in the right direction these approaches have focused their testing on typical POSIX metadata which is fully populated, low dimensional, and primarily numeric [2, 3]. By contrast, scientific data is high dimensional, heterogeneous, and very sparse [4]. These findings indicate that approaches such as naive row-major databases, and spatial trees scale poorly to scientific data as they waste space to store null values, have inflexible schemas, and have trouble with many to one values.

We propose an indexing technique called *On-demand Indexing*, which will create relevant indexes at search time if one does not already exist, while simultaneously providing query results. On-demand indexing will achieve scalability by only indexing searched for terms as well as utilizing a storage substrate better suited to high dimensional, heterogeneous, sparse data. Despite the simple approach of indexing everything that is searched for, we believe that like web search [1], file system search term frequency follows a power-law distribution. This means that a majority of searches will be the result of a few frequently searched terms. As a result the system would provide low latency, high precision, high recall search without the storage and computational overhead of indexing all data.

Architecture

The on-demand indexer will consist of three main components: the filter, the indexer, and the storage substrate. The ingest process will handle pre-processing of data. It will collect and store a table of which files contain which features, attributes, and content; this data is known as *filter data*. The indexer will create and manage indexes by using filter data to determine if new files must be parsed and indexed, or old index records pruned. Rather than naively processing all data when an index must be built, the indexer will check the filter data to know exactly which files contain relevant attributes and must be processed further. This will aid in scalability by eliminating the need for computationally expensive, memory intense brute force techniques. Both the filter data and the index data will be stored in a column store. Column stores group data on disk by column, offering a number of benefits for our uses such as a flexible schema, and better compression compared to typical row based approaches.

The Filter: As a file is written, it will be parsed by the filter for important data such as column headings, metadata fields, or even content markers, and noting which files contain these attributes. This data will be used to speed up subsequent indexing by reducing the set of files that must be parsed to create an index. Filter data will be collected by pluggable transducers specific to the data format as data is ingested. The filter data will be stored along side index data within a column store. In addition to collecting data on new files, the filter will be responsible for tracking file updates as well. When files are updated, the file system will notify the filter to remove filter data and invalidate index entries that are no longer accurate, as well as track the update times of files.

The Indexer: When a query is issued to the indexer it will first check to see if there is an index for the query terms. If one or more indexes *does not* exist it will use filter data to determine which files must be further processed to create the relevant indexes. Once a list of files has been narrowed via the filter data, the indexer will parse the relevant files and build the appropriate indexes.

If one or more indexes *does* exist at query time, the indexer will verify that they are up-to-date by first verifying that the filter data does not contain additional files to be processed for that index, as well as checking for

invalidated index entries. If there are no such entries the query is answered with index data. If the filter data contains new files to index they will be parsed and added to the index. If entries have been invalidated the query is answered, and the index pruned.

Storage Substrate: Informed by Parker-Wood *et al.*'s study of scientific data [4], the on-demand indexer will make use of a column store as storage substrate. Grouping data by column provides a number of benefits. First, new columns can easily be added to a table as needed. This allows for the table schema to change without the need for an administrator to curate or modify the database. Second, null values need not be explicitly stored, providing at least a 20% storage savings based on Parker-Wood's sparsity observations. Third, column stores are capable of dealing with high cardinality data, data that has multiple values per attribute, without explicitly managed schemas or human curation. Last, the column store, by grouping data by column, allows for in situ optimizations such as keeping the column sorted on disk, eliminating the need for an additional separate index to be constructed over that column.

Status

Currently the on-demand indexer is still in the initial stages of development. Apache's HBase has been selected as the column store to facilitate easy integration with Hadoop and HDFS in the future.

Currently we are working on the ingest process, which is nearing completion. Transducers have been built to handle csv, and XML data formats and there are plans to also accommodate the HDF and NetCDF scientific formats. Once the ingest process is finished the indexer will be built, and tested by using HBase's in-built query optimizer and search. As the project matures additional search strategies may be tested.

In addition to code development we are collecting test data and sample queries to evaluate the project on real world data and real world queries.

References

[1] BAEZA-YATES, R. Applications of web query mining. In *Proceedings of the 27th European conference on Advances in Information Retrieval Research* (Berlin, Heidelberg, 2005), ECIR'05, Springer-Verlag, pp. 7–22.

[2] LEUNG, A., ADAMS, I. F., AND MILLER, E. L. Magellan: A searchable metadata architecture for large-scale file systems. Tech. Rep. UCSC-SSRC-09-07, University of California, Santa Cruz, Nov. 2009.

[3] NAPS, J., MOKBEL, M., AND DU, D. Pantheon: Exascale file system search for scientific computing. In *Scientific and Statistical Database Management*, J. Bayard Cushing, J. French, and S. Bowers, Eds., Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, pp. 461–469.

[4] PARKER-WOOD, A., MADDEN, B., MCTHROW, M., AND LONG, D. D. E. Examining extended and scientific metadata for scalable index designs. Tech. Rep. UCSC-SSRC-12-07, University of California, Santa Cruz, Dec. 2012.