CouchFS: A High-Performance File System for Large Data Sets

Fangzhou Yao, Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
{yao6, rhc}@illinois.edu

***Abstract*—Numerous file systems have been implemented to meet the needs in today's big data era, however many of them require specific configurations or frameworks for data processing. This paper presents CouchFS, a POSIX-compliant distributed file system for large data sets. We build CouchFS on top of CouchDB, which grants us flexibility to handle semistructured data. Since a database has similar behaviors as a file system, and CouchDB provides a high customizable MapReduce view for indexing, CouchFS is able to achieve high-performance searching for both text and supported binary objects. This work compares search of Wikipedia data using CouchDB, PostgreSQL and Spotlight on HFS+ file system. We show our design of CouchFS and discuss future approaches to improve this file system.**

I. Introduction

In today's big data era, much of the information is semistructured, such as web pages, RSS feeds and tweets, where representations like HTML, XML and JSON are widely used. These types of data do not have a regular structure and the data might be incomplete. Traditional Relational Database Management Systems (RDBMS) are limited when data is irregular, and hence Lore was introduced as an Object Database (OODBMS) to address this concern [7]. However, the Object Exchange Model (OEM) presented in Lore does not have Document Type Definition (DTD) as XML, and hence this model might become unreliable due to the lack of validation.

Lore was not designed for big data, as it was designed more than a decade ago. Nowadays, the Hadoop Distributed File System (HDFS) provides a framework to analyze very large data sets using MapReduce model [8]. However, it requires knowledge of programming with Hadoop. Thus, people tend to use Pig Latin, a SQL style language, to process MapReduce jobs on Hadoop [9], but this approach still needs extra effort in configuring and learning this framework. People also use HBase, a database system built on top of HDFS, to manage big data. It features data compression, in-memory operation and Bloom filter. However, it is not as performant as HDFS in a MapReduce context, and can be 4 to 5 times slower [5].

Database systems have much in common with traditional file systems [12]. Oracle introduced Oracle Database Filesystem (DBFS), and one of its important objectives is to provide a transparent abstraction of a shared file system to users [10]. However, its Large Object (LOB) data model is slow in reading large text data and few such file systems are designed with the consideration of big data. Thus, we propose CouchFS, a high-performance distributed file system built on top of CouchDB for large data sets, by taking advantage of its MapReduce view for indexing [1]. We also use Filesystem in Userspace (FUSE) to make it POSIX-compliant [4].

The rest of this paper is organized as follows. We first analyze features provided by CouchDB, and discuss potential benefits by building a file system on top of it in Section II. Next, we show our design in Section III for this file system. In Section IV, we conclude our work and provide future approaches to improve this file system.

II. Why Do We Use CouchDB?

This section explains why we used CouchDB to design a file system with the illustration of its unique features and our benchmark results.

CouchDB is a NoSQL database. It leverages the self-describing JSON document to store data and its APIs are in a RESTful fashion. These properties make it flexible and naturally affinitive with semistructured data. CouchDB is also a distributed DBMS. Its proxy-based partitioning and clustering application provide us an abstraction that is necessary for a distributed file system [1]. Thus, we do not have to implement the distributed framework from the beginning.

We conducted our experiments on OS X 10.9.1 on a Mac with Intel Core i7 2.93GHz CPU and 24GB memory. The data set used in our experiments was a part of the dump of Wikipedia databases [6]. It consisted of 157,717 Wikipedia pages in one XML file. We simulated the text mining in big data by searching random words in these pages. Thus, we imported those pages to PostgreSQL and CouchDB, with attributes of `id`, `title` and `text`, while splitting the XML file into text files in HFS+ with native Spotlight indexing from OS X. We wrote our mapper function in Javascript for CouchDB to split the body text in each page into unique words, and used them as keys for indexing, where the values were the IDs and titles. We ran 10 rounds of different case-sensitive keyword searches and obtained our results as below.

As it is shown in Figure 1, CouchDB is drastically performant than other frameworks because of its customizable indexing with our mapper function. Since the view in CouchDB is essentially a collection of key-value pairs, gathering search results is fast. Figure 2 shows the number of results returned from searches. We normalized results from Spotlight as 100% and adjusted others' correspondingly, because the keyword *Hadoop* only derived 3 results, while the keyword *Illinois* returned more than 3,000 entities. The differences occur, because CouchDB and Spotlight use words as keys for searching, but the PostgreSQL checks matched substrings. Thus, there

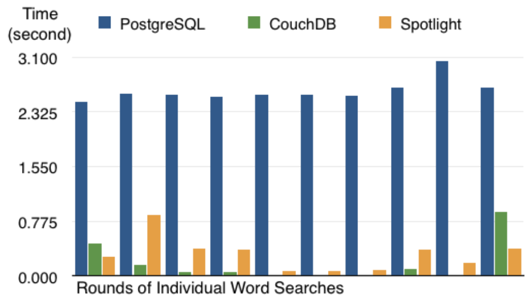

Fig. 1: Performance

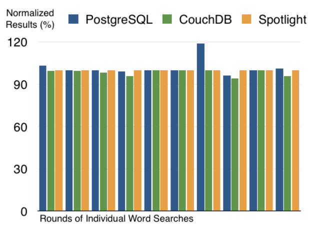

Fig. 2: Number of Results

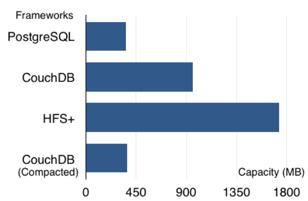

Fig. 3: Capacity

is a difference for keywords like *cat*, since pages containing *cats* are also included in results from PostgreSQL. Moreover, Spotlight also checks keywords in web links in pages by ignoring hyphens and slashes, but CouchDB does not. In Figure 3, CouchDB achieves less than half of its original size after compaction, which is only slightly greater than the data size in PostgreSQL.

III. Design

This section shows our design of CouchFS. We use FUSE to connect CouchFS to the userspace. There has been some implementations of CouchDB file systems with FUSE, like CouchFUSE [2]. However, it does not take the indexing or compaction into account and it only supports a single-level directory. In order to build a high-performance distributed file system for big data, we focus on the metadata and indexing.

The metadata for a file or directory is equivalent to an i-node in UNIX style file systems [11], which contains the file name, type and timestamps, as well as a unique `_id` attribute generated by CouchDB for every record. Besides, we keep a `directory` attribute in each record that points to its parent directory's `_id`, functioning for multi-level directory. We simply embed contents in text files into their metadata. However, since it has been proved that binary large objects (BLOB) might decrease the performance for a file system built on top of a database [12], we store BLOB on the disk as standalone attachments in CouchDB, and we keep track of them in metadata with an `_attachment` attribute. These attachments can be compressed by CouchDB. Furthermore, since binary files are difficult to search, we introduce file type providers in CouchFS to extract searchable text information from binary files, which makes this file system extensible to various file types. For instance, if we detect a PDF file, we will then call corresponding PDF type provider to generate its text and store it into the metadata. For MPEG-2 Audio Layer III (MP3) files, we derive their ID3 tags with our MP3 type provider, and keep them in their metadata as semistructures.

When a new attribute is introduced into CouchFS, we will create a new view for indexing. In CouchDB, an update on a view is incremental, such that it only reindexes the documents that have changed [1]. Thus, it takes much less time than the first time indexing and hence it is able to accomplish indexing for nearly real-time search. However, the first time indexing for 157,717 Wikipedia pages took about 3 hours. The process includes importing data into CouchDB and building the view with our mapper function. We wrote our mapper function in Javascript, but as it is pointed out, using native Erlang language to write functions could yield a 10-time faster performance [3]. In addition, in order to reduce the redundancy for more usable space, compaction can be enabled in CouchFS to reduce the size of data in the CPU idle time, though this solution may decrease the performance for write-intensive data. CouchFS also removes outdated views automatically.

IV. Conclusion and Future Work

In this paper, we analyzed CouchDB along with PostgreSQL and Spotlight running on HFS+. The results are significant to demonstrate that CouchDB can be a good choice to build a high-performance distributed file system for large data sets. We also showed our design of CouchFS by introducing our metadata mechanism and indexing schema.

In our future implementation, we will port our existing Javascript code to Erlang for faster indexing, and then we will benchmark our complete implementation with its performance, data capacity and network throughput. Besides, we did not write reducers to aggregate key-value pairs in views, and hence we will conduct experiments to discuss if a reducer will achieve better performance for CouchFS in searching.

References

[1] J. C. Anderson, J. Lehnardt and N. Slater, *CouchDB The Definitive Guide*, [Online]. Available:http://guide.couchdb.org/

[2] R. Narkis, *Couchfuse*, [Online]. Available: http://goo.gl/N8xAhZ

[3] J. Guerra, *Reduce View Indexing Time for CouchDB*, [Online]. Available: https://coderwall.com/p/odaeca

[4] M. Szeredi, *FUSE*, [Online]. Available: http://goo.gl/8B3c

[5] The Apache Software Foundation, *Apache HBase Performance Tuning*, [Online]. Available: http://hbase.apache.org/book/perf.hdfs.html

[6] Wikimedia Foundation, *Wikipedia Database Backup Dumps*, [Online]. Available: http://dumps.wikimedia.org/

[7] J. McHugh, S. Abiteboul, R. Goldman, D. Quass and J. Widom, *Lore: A Database Management System for Semistructured Data*, SIGMOD Record, Vol. 26(3), 1997.

[8] K. Shvachko, H. Kuang, S. Radia and R. Chansler, *The Hadoop Distributed File System*, 26th IEEE Symposium on Mass Storage Systems and Technologies, 2010.

[9] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, *Pig Latin: A Not-So-Foreign Language for Data Processing*, 2008 ACM SIGMOD international Conference on Management of Data, 2008.

[10] K Kunchithapadam, W. Zhang, A. Ganesh and N. Mukherjee, *Oracle Database Filesystem*, 2011 ACM SIGMOD International Conference on Management of Data, 2011.

[11] A. S. Tanenbaum, *Morden Operating Systems*, 3rd Edition, Pearson Prentice Hall, 2008.

[12] R. Sears, C. van Ingen and J. Gray, *To BLOB or not to BLOB: Large Object Storage in a Database or a Filesystem?*, Microsoft Research TechReport, MSR-TR-2006-45, 2006.