HadoopDB in Action: Building Real World Applications

Azza Abouzied, Kamil Bajda-Pawlikowski, Jiewen Huang,
Daniel J. Abadi, Avi Silberschatz
Yale University
{azza, kbajda, huang-jiewen, dna, avi}@cs.yale.edu

ABSTRACT

HadoopDB is a hybrid of MapReduce and DBMS technologies, designed to meet the growing demand of analyzing massive datasets on very large clusters of machines. Our previous work has shown that HadoopDB approaches parallel databases in performance and still yields the scalability and fault tolerance of MapReduce-based systems. In this demonstration, we focus on HadoopDB's flexible architecture and versatility with two real world application scenarios: a semantic web data application for protein sequence analysis and a business data warehousing application based on TPC-H. The demonstration offers a thorough walk-through of how to easily build applications on top of HadoopDB.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Design, Performance

1. INTRODUCTION

As data volumes increase, so do requirements for efficiently extracting value from data sets. With more and more businesses reporting petabyte-sized data warehouses, the system of choice for managing and analyzing massive amounts of the data ("Big Data") will be the one that (i) provides high performance, (ii) scales over clusters of thousands of heterogeneous machines and (iii) is **versatile**.

Versatility refers to the adaptability of a system to analytical queries of varying complexity. Versatility also refers to system flexibility in terms of query languages supported, and control in terms of data preparation, placement and management.

Of these features, recent research in the data management community has focused on evaluating the performance and scalability of two major data analysis technologies: MapReduce (in particular, the Hadoop implementation) and parallel databases [8, 5].

The inherent architectural and conceptual differences between these technologies has led to a variety of interface-level hybrids such as Hive [9] and Pig [7] and systems-level hybrids such as HadoopDB [5], all of which attempt to bring the best of both worlds into one system. HadoopDB combines the scalability features of Hadoop with the high performance of database systems for structured data analysis. HadoopDB is an open-source project [1].

This demonstration focuses on HadoopDB's versatility. We answer the question, *"How does one build real world applications with HadoopDB?"*, by developing two applications: a semantic web application that provides biological data analysis of protein sequences, and a classical business data warehouse. In previous work, we also employed HadoopDB in web-log analysis [5].

The demonstration goes over the design of the applications in great detail, from data preparation to query interface setup for data analysts. Our target audience therefore includes system architects, application developers, and data administrators, as well as data analysts and researchers dealing with large data.

This paper begins with a description of HadoopDB's architecture and how it integrates with the traditional multi-tier architecture of current software applications. In Section 3, we describe the two applications and highlight our motivations for using these particular cases as a representative sample of applications that could benefit from HadoopDB. Finally, we describe the demonstration scenario in Section 4.

2. ARCHITECTURE AND DESIGN

The architecture of HadoopDB is illustrated in Figure 1. HadoopDB connects multiple independent single-node database systems deployed across a cluster, using Hadoop as a coordination layer that manages task execution and network communication. To maximize performance, as much of the data processing as possible is pushed into the underlying database systems. HadoopDB therefore resembles a shared-nothing parallel database with Hadoop providing services necessary for scalability over thousands of nodes, such as runtime task scheduling.

The key components of HadoopDB include:

1. A *Database Connector* that connects Hadoop with the single-node database systems.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

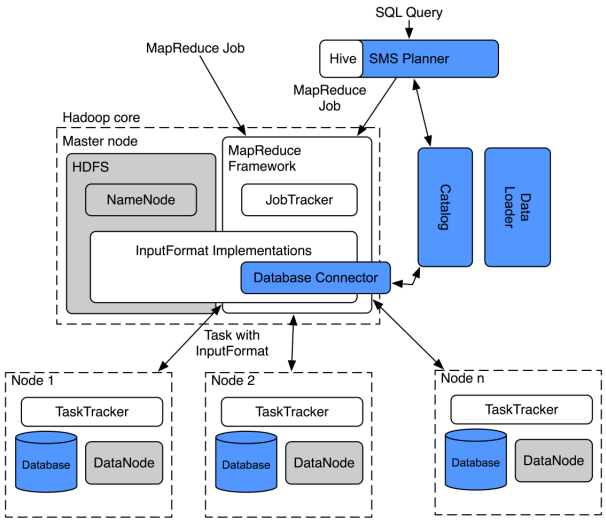

Figure 1: The HadoopDB Architecture

2. A *Data Loader* which partitions data and manages parallel loading of data into the database systems.

3. A *Catalog* which tracks locations of different data chunks, including those replicated across multiple nodes.

4. The *SQL-MapReduce-SQL (SMS)* planner which extends Hive [9] to provide a SQL interface to HadoopDB.

See our previous work [5] for more details on the HadoopDB architecture.

HadoopDB supports any JDBC-compliant database server as an underlying DBMS layer, giving application designers a lot of flexibility to choose the most efficient database technology for a given application. In the original HadoopDB paper [5] we evaluated our prototype with PostgreSQL. In this demonstration, we utilize a column-oriented database instead.

Applications built on top of HadoopDB generally use the three-tier architecture commonly found in today's software applications. In a three-tier architecture, the *data tier* stores and retrieves data, the *business logic tier* performs data processing and the *presentation tier* provides an interface to users. In the classic three-tier architecture, the business logic tier performs much of the necessary computation on the data retrieved from the data tier. However, for typical HadoopDB applications, given the size of data and the amount and complexity of data analysis tasks, this arrangement is no longer optimal. HadoopDB therefore pushes computation closer to data (into the data tier) to achieve maximum parallelization in a multi-node cluster. The business logic layer becomes a simple application server, providing only query translation and connectivity services between the presentation tier and the data tier.

From the application perspective, HadoopDB is a black box. The complexity of the data tier and its parallel nature is hidden from the application developer: applications interact with HadoopDB through a JDBC interface which supports submission of SQL queries as well as invocation of stored procedures implemented using HadoopDB's API.

3. EXAMPLE APPLICATIONS

We demonstrate the versatility of HadoopDB using two different applications: a semantic web/biological data analysis application and a business data warehousing application.

Both applications involve complex analytical queries involving multiple relations. In each case, analysts expect small response times for their queries. Moreover, the selected data sets do not simplify the problem of data preparation. Instead, they highlight the importance and complexity of data preparation. In our demonstration, we describe in detail how HadoopDB allows for query optimization through appropriate data preparation techniques that are not possible in Hadoop alone. For example, through careful use of copartitioning multiple tables, HadoopDB can push joins into the single-node database systems, leveraging the high performance join implementations within these systems. Hadoop's join implementation is substantially slower (especially if the join is performed in the Reduce phase). Careful indexing can also significantly improve performance.

3.1 Semantic Web — Biological Data Analysis

The semantic web is an effort by the W3C to enable integration and sharing of data across different applications and organizations. Unlike the relational data model, the semantic web uses the Resource Description Framework [10] (RDF) data model. RDF data consist of statements about resources and can be queried using the SPARQL query language [11].

Abadi et al. have demonstrated that semantic web data can be stored and queried very efficiently in a column-oriented database [4]. As the size of the semantic web continues to grow, however, single-node data management systems are quickly becoming inadequate. Effective parallelization of semantic web querying and inference is critical to the survival of semantic web technology.

HadoopDB, in conjunction with a column-oriented database, is a promising solution for supporting efficient and scalable semantic web applications. We utilize the UniProt [3] comprehensive catalog of protein sequence, function, and annotation data to validate this hypothesis. The data set contains over 600 million statements and continues to grow as more proteins are sequenced.

Our implementation allows analysts to specify queries in SPARQL. For example, to find all proteins whose existence in the 'Human' organism is uncertain, the following query can be submitted:

```
SELECT ?prot ?name
WHERE {
  ?prot <organism> <9606> ;   # 9606 means Human
        <name> ?name ;
        <existence> <Uncertain_Existence> .
      }
```

Figure 2 illustrates the overall application architecture. The presentation layer consists of a web-based interface where analysts specify queries and view results. The logic layer consists of a SPARQL to SQL conversion tool [6]. Finally, the data tier consists of HadoopDB maintaining and processing the full data set. The logic and data layer communicate through JDBC.

We demonstrate how the data administrator should prepare the dataset. The analyst, however, is shielded from the complexity of the actual implementation of the RDF stor-

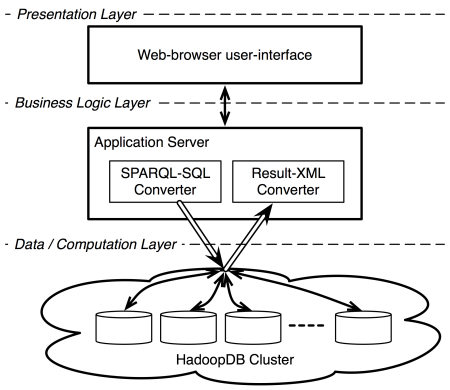

Figure 2: The UniProt Application Architecture

age layer. A final motivation for choosing this application to demonstrate HadoopDB is that by making our implementation publicly available, we can enhance the infrastructure for life sciences research.

3.2 Business Data Warehousing

Business data warehousing is a natural target application for HadoopDB. Due to superior fault-tolerance, HadoopDB scales to very large data sets much more gracefully than existing parallel databases [5]. Moreover, common business data warehousing workloads are read-mostly and involve analytical queries over a complex schema. To achieve good query performance, the dataset requires significant preparation through data partitioning and replication to optimize for join queries.

We use the data and queries from the TPC-H benchmark [2]. TPC-H models a global distribution company. It contains information on suppliers, products, customers and orders in different regions. The benchmark consists of various complex business analysis queries that aid with decision making, pricing and revenue management.

For example, the following query asks for the ten highest-revenue unshipped orders:

```
SELECT l_orderkey, o_orderdate, o_shippriority,
   SUM(l_extendedprice * (1 - l_discount)) AS revenue
FROM customer, orders, lineitem
WHERE c_mktsegment = 'BUILDING' AND c_custkey = o_custkey
  AND l_orderkey = o_orderkey AND o_orderdate < '1995-03-15'
  AND l_shipdate > '1995-03-15'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue DESC, o_orderdate LIMIT 10;
```

4. DEMONSTRATION SCENARIO

The audience is invited to query both data sets through HadoopDB. The data sets are located in a remote cluster. To allow multiple users to interact concurrently with the cluster, we provide two client machines that connect to the clusters. In case of limited connectivity in the demo room, a local two-machine mini-cluster housing subsets of both data sets is also available. Each instance operates independently of the other.

The user first selects from a web-browser interface the data set they wish to query. An animation of the behind-the-scenes data preparation and loading is then presented. In the case of TPC-H, the animation provides details on the partitioning scheme, the interaction between the loader and catalog components, and a summary of the configuration parameters. The UniProt animation also includes details on the tools used for data conversion from RDF to relational form. These presentations provide our audience with an idea of the effort required for data preparation in HadoopDB.

The interface then invites the user to select and parametrize a query to execute from sets of predefined queries for both applications. The user can then monitor the progress of query execution before finally being presented with the results in a tabular form for TPC-H or XML form for UniProt. In addition, we demonstrate HadoopDB's fault-tolerance with the introduction of a node failure.

For a subset of the predefined queries, as the query executes in the background, an animation of the flow of data and control through the HadoopDB system is simultaneously presented, highlighting which parts of the query execution are run in parallel.

5. ACKNOWLEDGEMENTS

We thank Alexander Thomson for his suggestions regarding this demonstration.

This work was sponsored by the NSF under grants IIS-0845643 and IIS-0844480. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

6. REFERENCES

[1] HadoopDB Project. http://hadoopdb.sourceforge.net.

[2] TPC-H. http://www.tpc.org/tpch/.

[3] Universal Protein Resource. http://www.uniprot.org/.

[4] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. SW-Store: A Vertically Partitioned DBMS for Semantic Web Data Management. VLDB Journal, 18(2), April 2009.

[5] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In *VLDB*, 2009.

[6] K. Bajda-Pawlikowski. Querying RDF data stored in DBMS: SPARQL to SQL Conversion. *Yale CS Technical Report TR-1409*, 2008.

[7] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proc. of SIGMOD*, 2008.

[8] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. R. Madden, and M. Stonebraker. A Comparison of Approaches to Large Scale Data Analysis. In *SIGMOD*, 2009.

[9] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murth. Hive — A Petabyte Scale Data Warehouse Using Hadoop. In *Proc. of ICDE*, 2010.

[10] W3C. Resource Description Framework. http://www.w3.org/RDF/.

[11] W3C. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/.