Criteria to Compare Cloud Computing with Current Database Technology

Jean-Daniel Cryans, Alain April, and Alain Abran

École de Technologie Supérieure, 1100 rue Notre-Dame Ouest
Montréal, Québec, Canada
jean-daniel.cryans.1@ens.etsmtl.ca
{alain.april,alain.abran}@etsmtl.ca

Abstract. After Google published their first paper on their software infrastructure in October 2003, the open-source community quickly began working on similar free solutions. Yahoo! is now able to process terabytes of data daily using Hadoop, which is a scalable distributed file system and an open-source implementation of Google's MapReduce. HBase, a distributed database that uses Hadoop, enables the reliable storage of structured data, just like Google's Bigtable which powers applications like Google Maps and Google Analytics, to name only two. Many companies are tempted to use these technologies, but it is currently difficult to compare today's systems with systems built on top of HBase. This paper presents this new technology and, a list of proposed comparison elements to existing database technology as well as proposed comparison assessment criteria.

Keywords: Cloud Computing, Bigtable, HBase, Hadoop.

1 Introduction

After Google published their first paper on their proprietary software infrastructure in October 2003 [1], the open-source community quickly began working on similar open-source solutions. Today, Yahoo! and many other companies are able to process terabytes of data daily using the open-source solution called the Hadoop [2] framework. HBase [3], a sub project of Hadoop, is a distributed database built on the same specifications as Google's Bigtable [4] which powers applications like Google Maps and Google Analytics, to name only two. Reasons to try to learn and understand this technology include eliminating licensing costs, achieving scalability and better control over the performance characteristics of the applications. When it takes a few months for a typical organization to choose, order, install, and set up a few servers, it is already too late when your Web site is growing by 30 million pages per day. This situation was faced by YouTube during 2006 [5], and they turned to the Bigtable technology to solve their issues when they were bought by Google.

Installing and using HBase is not something we learn to do in school; in other words, these tasks are not, at first glance, intuitive. At the bottom of the infrastructure illustrated in Fig. 1, there is a distributed file system designed to scale to thousands of

R. Dumke et al. (Eds.): IWSM / MetriKon / Mensura 2008, LNCS 5338, pp. 114–126, 2008.

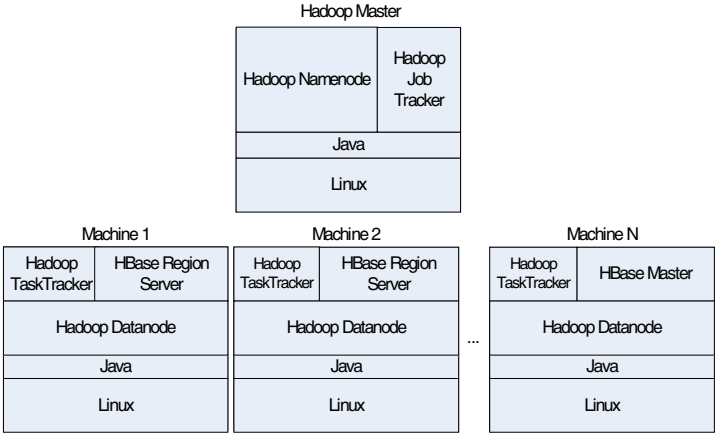

Fig. 1. HBase infrastructure (master and region servers)

machines. This is the Google File System [6] at Google and the Hadoop Distributed File System in the open-source community. They are both tolerant to machine failure and replicate data which can be counted in petabytes. To query such huge datasets, Google invented a new programming model called MapReduce [7], the idea behind it being that processing data in a distributed environment always involves the same two basic steps: 1) mapping the data needed; and 2) aggregating those data. Now, trying to move such datasets on a network would easily saturate its capacity. So, unlike supercomputing, processing is performed where the data are located -- that is, on each node -- and must be managed by a master scheduler and a number of scheduler slaves. However, a file system alone is inefficient at handling structured data, which is why Google created Bigtable, a distributed database that leverages the distributed file system. Bigtable, and its open-source equivalent, HBase, offer a simple and dynamic data model that is not relational.

Using this infrastructure implies better availability, as well as scalability and new tools to process huge amounts of data, making it is useful for large corporations, but also for startups, whose aim is to develop a large amount of traffic. Those who wish to migrate from a typical open-source relational database management system (RDBMS) to HBase currently have no tools or methodologies with which to compare the available systems. Our paper is aimed at helping these companies by presenting a list of comparison elements and demonstrating how they translate into assessment criteria. This research was carried out in parallel with a project in which our lab was working on migrating a system based on PostgreSQL for a Canadian telephone company. In section 2, we describe the Hadoop distributed file system, MapReduce, and HBase function on the conceptual level, and provide examples. In section 3, the comparison elements are defined. In section 4, we describe how the technique is used to transform comparison elements into assessment criteria. Finally, we conclude the paper with a summary and a description of future work.

2 An overview of HBase and Its Underlying Infrastructure

2.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a distributed file system for large data-intensive applications, and is designed to run on commodity hardware. It is similar in many ways to existing distributed file systems, most notably the Google File System (GFS), and shares the same goals, such as performance, reliability, and availability. The HDFS initially served as the infrastructure for the Apache Nutch [10] Web search engine project, and is now part of the Apache Hadoop Core project. Its core developers come from Yahoo!'s Grid Team, which currently has the largest HDFS cluster (a group of computers working on a common computation) in production; it runs on more than 10,000 processors and stores over 5 petabytes of information [11].

Assumptions and Goals

- The clusters are built from commodity machines, the components of which often fail. The file system must detect faults from which it should automatically and quickly recover.
- The applications that need to access the data stored in the HDFS need streaming access. They are not general-purpose applications that connect to POSIX-compliant file systems. They rely on the HDFS for batch processing and not for interactive use by users.
- The applications that run on the HDFS have medium to large datasets. The typical file stored is gigabytes to terabytes in size. The HDFS supports small files, but is not optimized for them.
- Moving computations is cheaper than moving data; thus, running computations is much more efficient if executed near the data on which the computation operates. This is especially true with gigabytes of data. It prevents saturation of the network and increases the aggregated throughput of the system.
- The applications that run on the HDFS come from different platforms, so it is designed to be portable from one platform to another. This facilitates widespread adoption of the HDFS as the distributed file system of choice for a large set of applications.

Architecture

The HDFS is based on master/slave replication. As illustrated in Fig. 2, the master server is called a Namenode, and it both manages the file system namespace and regulates client access. The other nodes are called Datanodes, and they manage storage attached to the nodes on which they run. That storage is composed of files which are split into blocks, and it is the duty of the Namenodes to determine the mapping of the blocks to the Datanodes. Although the HDFS is not POSIX-compliant, its file system still supports the usual operations of creating, deleting, opening closing, reading, and writing files.

The typical HDFS cluster is built on commodity machines which run a GNU/Linux operating system. But since the HDFS is written in the Java language, any machine supporting Java can run a Namenode or a Datanode. It is recommended that the

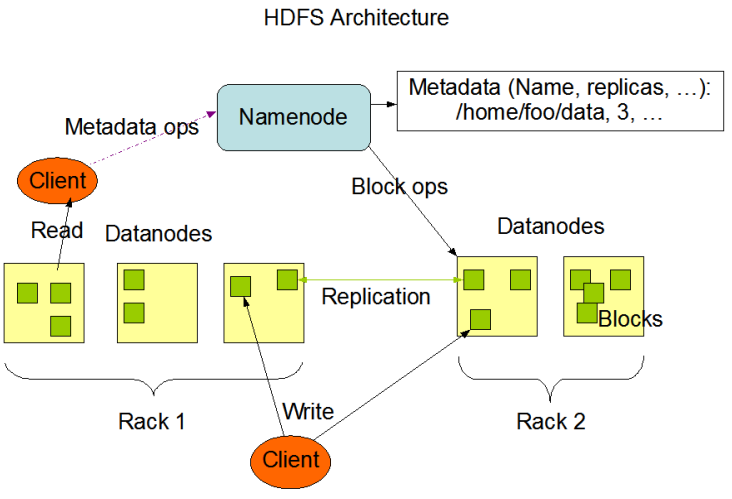

Fig. 2. The HDFS architecture. The file blocks are distributed among the Datanodes and monitored by the Namenode. The clients read and write directly from the Datanodes [12].

machine that hosts the Namenode be dedicated to that Namenode, and does not host a Datanode for main memory considerations. The other machines host only one Datanode, as running multiple instances greatly reduces the reliability of the system.

Moreover, having only one Namenode greatly simplifies the architecture of the system. In order to ensure that a bottleneck is not created, clients never read or write files through the Namenode. Instead, they ask the Namenode which Datanodes are hosting the blocks associated with the files they need.

2.2 MapReduce

The MapReduce programming model was invented by Google for processing large datasets across hundreds or thousands of machines. The software framework hides the details of computation parallelization, data distribution, and failure handling, and lets users specify a map function which transforms a set of key/value pairs into a set of intermediate key/value pairs and a reduce function that merges all intermediate values associated with the same intermediate key. The Apache Hadoop Core project contains a Java implementation of the MapReduce specification.

Programming Model

The Map function is written by the user and takes as input a key/value pair and outputs an intermediate key/value pair set. The Reduce function is also written by the user and accepts an intermediate key and a set of values for that key. The values are merged in such a way that only an output value of zero or one is produced per reducer.

The Map function will output each word in a document with an associated count of occurrences which, in this function, will always be '1'. The Reduce function will then, for each word outputted, sum its occurrences and output that figure.

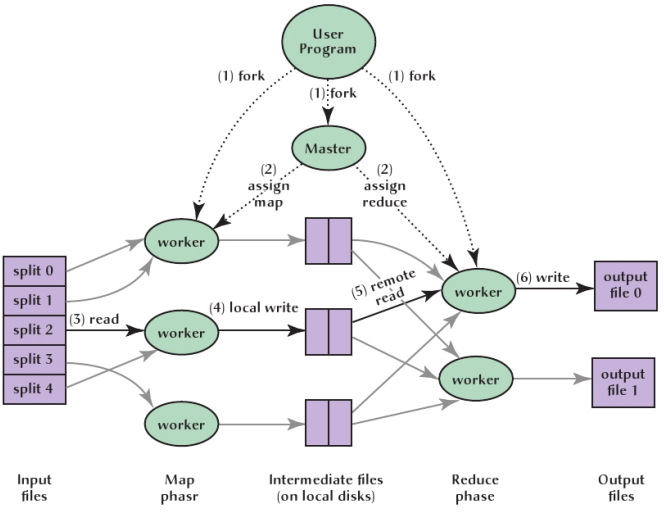

Fig. 3. The MapReduce execution flow in Google's implementation [7]

Implementation

The Hadoop MapReduce implementation [22] greatly resembles that of Google, since both are designed to handle petabytes of data daily. The typical cluster consists of the same types of machine used for the HDFS, and uses it as its distributed file system. The jobs, a set of Map and Reduce functions, are submitted to a scheduling system to the available machines.

In summary, the execution flow proceeds in the following sequence (numbering refers to the numbers in Fig. 3):

1. The Map functions are started for each input file split on its hosting machines.
2. A special node, the JobTracker (or Google's Master), assigns a Map or a Reduce task to each idle TaskTracker (or Google's Worker).
3. The TaskTracker then reads the block it was assigned, and parses each key/value pair out of the input data and passes it on to the Map function.
4. The Map outputs are then sorted and partitioned per Reduce function. A Combine function is an optional which can be used to aggregate the outputs of each Map of each machine, thereby limiting data transfer over the network.
5. When a TaskTracker responsible for a Reduce is notified that mapped data are available, it reads its content via HTTP. As the data are retrieved, they are sorted by the intermediate key.
6. Once the intermediate keys are sorted, the Reduce TaskTracker iterates over each unique key and passes it along with its set of intermediate values to the Reduce function. The output is appended to a final output file for this Reduce partition.

Usage

Google uses MapReduce in a variety of internal applications, and states that, in September 2007, 11,081 machines were used in MapReduce jobs [13].

Yahoo! uses Hadoop in their Web search, and the output of their MapReduce tasks is over 300 terabytes, compressed [11].

Rackspace uses Hadoop to parse its logs generated from the machines of their datacenters to collect statistics (such as spam counts) from their mail system [14].

Facebook uses Hadoop in a 320-machine cluster to store internal log and dimension data sources, as well as for reporting, analytics, and machine learning [15].

2.3 HBase

HBase is an open-source, distributed, column-oriented, multi-dimensional, high-availability, high-performance storage technology written in Java and inspired by Google's Bigtable. Just as Bigtable leverages the distributed data storage provided by the Google File System (GFS), HBase is aimed at providing Bigtable-like capabilities on top of the HDFS. HBase has been a part of the Apache Hadoop project since February 2008, and is used in production environments.

Motivations

Scaling out a typical relational database often begins with replication. For example, YouTube [5] first used MySQL master-slave replication, but eventually arrived at a point where the writes used all the capacity of the slaves. Like many organizations, they tried partitioning their tables into shards, so that sets of machines hosting the various databases were optimized for their tasks. Flicker [19] did the same thing with their shards, and even duplicated some data since the comments table was linked to their user shard and to their images shard. Soon, after going through many difficult steps to scale the architecture, they found that the first relational solution became denormalized and harder to maintain. Tools like Pyshards [17], aimed at easing shard management, were developed, but they do not obviate the need for partitioning a shard if it becomes too big.

Goals

To quote the authors of HBase, "The HBase project is for those whose yearly Oracle license fees approach the GNP of a small country or whose MySQL install is starting to buckle because tables have a few BLOB columns and the row count is heading north of a couple of million rows." [18] HBase is designed to leverage the HDFS to reliably store terabytes of sparse structured data. It should be as usable in real-time applications, where low latency is the priority, as in batch jobs, where higher throughput is important. HBase should not provide a relational model, but instead consist of a simple data model which supports dynamic modifications of data layout.

Data Model

HBase, like Bigtable, is a sparse, distributed, and persistent multi-dimensional sorted map which is indexed by a row key, a column key, and a timestamp, and the value is an uninterpreted array of bytes, as illustrated in Fig. 4. The column keys reside in column families, which are the same for each row and have the syntax "family:qualifier".

$$(row{:}string, column{:}string, time{:}int64) \rightarrow string$$

Fig. 4. Representation of HBase indexation of the values [4]

This map is persistent because it is stored in HDFS, which also makes it distributed. It is sorted because the row keys, not their values, are maintained in a lexicographical order in the cluster. It is multi-dimensional because the value of each row's key/column key pair has a timestamp; it is sparse because the column keys in column families are not necessarily the same across rows; and it is distributed because its components reside on many machines.

The database schema in HBase and Bigtable is very different from the relational schema. First, there are no joins between tables. For example, a blog would have three tables: blogs have many comments and many users. A solution for the blog/comment relation would be to have column families for each comment property, and the column key would be the comment ID. If the comments have properties that do not need to be accessed with their blog entries (such as the Web browser that was used), a second table can be created containing all the information. Since the system runs on commodity hardware, the disks are cheap and space is abundant.

The schema also has only one index, the row key. Unlike the RDBMS, the developers cannot use the equivalent of the WHERE clause. By design, all access methods in HBase are either gets or scans over the row key, so that a query cannot slow the whole system down. Since HBase sits on the Hadoop framework, MapReduce can be used instead to generate index tables.

HBase column families can be configured to support different types of access patterns. A maximum number of cell timestamps can be specified, as well as a time to live (TTL). By default, HBase returns the latest cell version available, keeps only one version, and has no TTL. A family can also be set to stay in-memory, yielding better performance at the expense of consuming the main memory. Also, since the data in a family are typically of the same nature, a compression level can be configured. For example, compressing a family that stores many versions of big strings which are not read often will leave more space on the disk for other, more important information.

Architecture

An HBase cluster is composed of individual Master and Region Servers which can be counted in the hundreds. The Master's job is to monitor the load and to coordinate the Region Servers. Each Region Server hosts a number of Regions, which it stores in the HDFS. A Region is composed of sorted rows from a table, so that all table rows are contained in a set of Regions. HBase relies on Zookeeper [16], which, like Bigtable, relies on Chuggy[8], to ensure that there is always one Master at any time, to store the bootstrap location of HBase data and to discover the Region servers.

The way the architecture works is based on the fact that the rows are ordered by row key. A three-level hierarchy, similar to a B+ tree, is used to store the Region locations. The first level is a file stored in Zookeeper that contains the location of the ROOT Region that the Master reads during its startup. That Region then gives the location of all the other META table's Regions, which in turn give the location of all the user Regions. So, when an application requests a particular row, it first reads the META table to figure out in which Region the row is located and then directly contacts the server hosting the Region to retrieve the row. There are many caching levels, but these are not discussed in this paper. The writes are processed in a similar fashion. Instead of figuring out which Region hosts the row, it will figure out which Region *should* host it, once it is sorted. If a Region becomes too big according to a configured threshold, it will be split into two.

Accessibility
HBase is accessible from its Java Application Programming Interface (API) which offers Create, Read, Update, and Delete (CRUD) operations as well as cluster management tools. The following program code demonstrates its use. The first operation shows how to open an access to an HBase table and fetch a value using a row and a column key. The second operation shows how to open a transaction on a row, insert a new value under a column key, and delete another value.

```
(1)HTable table = new HTable("myTable");
   byte[] valueBytes = table.get("myRow",
      new Text("myColumnFamily:columnQualifier1"));
   String valueStr = new String(valueBytes);

(2)long lockId = table.startUpdate("myRow");
   table.put(lockId, "myColumnFamily:columnQualifier1",
      "columnQualifier1 value!");
   table.delete(lockId,
"myColumnFamily:cellIWantDeleted");
   table.commit(lockId);
```

HBase can also be accessed via a REST gateway, which supports the statements GET and PUT which, when executed, return data in XML format. Another way to communicate with HBase is via a Thrift [20] gateway. Thrift is a framework enabling cross-language Remote Procedure Calls (RPCs) and was developed by Facebook. This means that most of the APIs can be accessed by the following programming languages: C++, Ruby, Python, and PHP.

3 Comparison Elements

We investigate a case study where an existing system, using RDBMS technology, has been migrated to the HBASE technology. In our lab, both systems are operating in a controlled environment. Our goal here is to compare the two systems and draw some conclusions on the usefulness of this new technology. In comparing the systems, many factors need to be taken into account. Initially, it appeared that the comparison would be difficult, as some elements do not vary, while others are very different. Below is a discussion on the first set of comparative elements we would like to assess:

- **Software architecture:** The software architecture could remain the same for each implementation. Typically, modern applications will have a presentation layer which communicates with a business logic layer, which in turn communicates with a database layer. In migrating to HBASE, only the database layer should need adaptation.
- **Hardware:** The hardware in the two implementations was definitively different. Using an RDBMS, the typical machines for a website with a sizeable traffic volume will be composed of as many processors and as much memory as can be fitted onto a motherboard. These components will also be server-class, since the consequences of a machine failure on availability are dire. This equipment is also expensive. The disks are organized in the RAID (redundant array of independent disk) technology that maximizes writing and

reading speed, along with replication. That equipment is installed in a master-slave scheme recommended by the manufacturer of the RDBMS. If the cost of replicating the write operations becomes too great, the database will be partitioned into shards.

In contrast, the equipment typically used for HBase is PC-class, so they are not very expensive. The recommended components are a dual or a quad processor with as many gigabytes of memory as possible and two hard drives of at least 250 gigabytes. All these machines will work together in a cluster which can consist of from 1 to 500 machines.

- **Operating system:** The majority of RDBMSs can be deployed on many operating systems, but HBase is currently only supported on the Linux distributions. This has to be taken into account if the RDBMS is not hosted on Linux.
- **Data structure:** The two data structures are very different. It is assumed that the reader is familiar with the relational schema, so it will not be described here. The Bigtable/HBase schema is described in section 2. The main differences are that all the relational primitives have been relaxed to allow a more dynamic table schema, and that there is no full scan available in HBase. This may seem restrictive from the point of view of a developer of small systems, but the data-intensive expert will recognize that this is also not doable in a relational system because the joins cannot all be performed in memory (unless the database was denormalized, which requires much more disk space).
- **Data manipulation:** Data manipulation is very rich and mature for online operations in an RDBMS and very restricted in HBase. It is assumed that the reader is familiar with SQL, so it will not be described here. The data manipulation for HBase is described in section 2. The biggest advantage of using HBase is that it is enabled to work with MapReduce as a source or as a sink, or both. Instead of performing the equivalent of a JOIN at runtime, it is more convenient in HBase to build an index table. For example, Bigtable's paper describes the way to generate the many reports in Google Analytics, which is to run a nightly MapReduce job which takes a table of clicks from each website as a source and an aggregate table as a sink.
- **Means to scale:** Scaling an RDBMS first means upgrading each machine, and, since these machines are expensive, the whole process is even more so. This method has a physical limit, which is the current state of the technology, and, unless there are redundant databases with custom load balancing, a downtime has to be scheduled. Scaling can also be performed with replication, so that each slave can handle the read operations while the master receives the writes. It is a method which works until the replication of each write on each server generates a bottleneck. This problem is solved by sharding the database, but an additional layer of software is needed to handle the shards. In any case, all these operations are very expensive.

  Scaling HBase is simply done by adding more machines. For each new machine, this is a matter of installing the operating system and the software, and then starting the Hadoop and HBase processes. The Datanode/Region servers will contact their respective master node, and, in return, will begin redistributing the load. The software that runs in the cloud will not be impacted.

- **Where hardware reliability is handled:** Hardware reliability is a big issue with any RDBMS, since these systems are designed to work on only one machine initially. This is why expensive solutions have to be designed to ensure that a machine failure will not take the whole system down. Having server-class components provides lower mean time between failure (MBTF), but it does not shield the system from a defaulting motherboard, which is why failover mechanisms are designed. These mechanisms can, for example, hold off the writes in case the master machine in a replication scheme dies to give system administrators time to modify the DNS to point to a new master. If a slave dies, a big part of the throughput goes down because of the small number of machines.

  In HBase, reliability is handled at the file system and database levels. If any slave machine dies, only a small part of it goes down, and this is considered "normal", given the number of machines. For example, bigger clusters like the ones they have at Yahoo! are able to sustain losing a whole rack.
- **How many systems are using the system:** A very important feature of cloud computing is that a single cloud can hold many systems, even if they have different quality requirements. It is economically wise to aggregate systems in a single cluster, because then every system will benefit from a newly added machine. In contrast, with the RDBMS, there are different databases for different applications, since scaling is difficult and expensive. When taking measures, one has to make sure to clearly define what in the cloud can be "given" to the software that is being compared to an RDBMS implementation.

This research was carried out in parallel with a project in which an existing system based on PostgreSQL had to be migrated to a system based on HBase. While choosing the hardware, the setup followed was the one that Google uses, as described in the Bigtable paper [4], and it was very different from that already in use. For example, the typical machine for HBase had dual processors, while those used in the source system were dual quad processors. To minimize the differences, the same operating system was kept, the system's architecture was not changed, and the cluster

Table 1. Comparison of elements proposed in the case study

Comparison element	PostgreSQL implementation	HBase implementation
Software Architecture	Three-tier	Three-tier
Hardware	Few, expensive	Scores, commodity
Operating System	Cent OS	Cent OS
Data structure	Relational tables	Bigtable-like structures
Data manipulation	ORM	HBase client API, MapReduce
Means to scale	Expensive	Inexpensive
Where hardware reliability is handled	Custom solution	HBase and Hadoop
How many systems are using it	One	One

was only used by the target system. By using Hadoop and HBase, the scalability and back end reliability requirements were easily met, so there was no need to keep the custom solution to handle them. Since the source system was in Java, the HBase client API was used directly. Table 1 summarizes the differences identified in each element during the research.

4 Transforming Comparison Elements into Comparison Criteria

To transform the comparison elements into criteria for a comparison assessment, a two-step technique was used. The process is presented below:

1. Identify impact: The impact that a comparison element has when it is not the same in the two implementations will lead to different measurement results.
2. Identify the direct effects on quality: Each comparison element has different effects on internal and external quality, as defined in the ISO/IEC 9126 models of software product quality [23].

Table 2 illustrates an application of this technique to derive assessment criteria for comparison.

Table 2. Assessment criteria mapping

Comparison element	Impact	Related ISO 9126 quality sub-characteristics
Software Architecture	Changing the architecture implies completely different quality attributes that need to be evaluated.	All criteria
Hardware	Commodity hardware is less reliable and, on its own, performs poorly.	Fault tolerance Time behavior Scalability
Operating System	If the source operating system is not Linux, its efficiency and maturity are different.	Fault tolerance Time behavior
	Restricting to Linux makes the system less adaptable.	Resource behavior Adaptability
Data structure	The target data structure is easy to change and provides constant performance.	Time behavior Changeability
	This data structure is not relational and not taught in classes.	Replaceability Adaptability
Data manipulation	The target data manipulation provides limited functionalities for online processing, but is excellent for offline processing.	Time behavior Analyzability Changeability
	MapReduce is unknown to most developers.	Replaceability
Means to scale	Scalability is built into Hbase, provides for easier reactions to new specifications, and does not require the system to be shut down.	Scalability Stability Adaptability
Where hardware reliability is handled	Reliability is also built in, so there is no need for a custom layer to provide it. The system is simplified.	Reliability Analyzability Changeability
How many systems are using it	The target system may perform badly if other systems are heavy users of the resources.	Time behavior

In Table 2, each row constitutes an assessment criterion which helps in comparing the quality of two systems. As described, HBase use may negatively affect the maintainability of the system, since its current and future developers may not be experts in Hadoop and HBase. At the same time, it provides a system that does not need to handle database failures. Moreover, it enables faster processing of large datasets and can scale according to changing requirements.

5 Future Work and Summary

While new technologies have been developed in the last few years to address the scalability and reliability problems inherent in data-intensive systems, little has been done to validate the quality of the new systems relative to the older ones. Based on our work in migrating a system that used PostgreSQL to one that used HBase, a list of comparison elements was identified and translated into assessment criteria. We do not claim that the list is exhaustive, and more research should be done to verify and validate the criteria it contains.

Future work is aimed at developing a process to measure the scalability and performance criteria. The only measures published for Bigtable-like databases to date are presented in Google's original paper [4]. Google used N machines and took measures as N varied while N clients were generating a load. The following measures were reported: random reads, random reads in memory, random writes, sequential reads, sequential writes, and scans for 1, 20, 250, and 500 machines. Results were impacted by the fact that the machines were also used by other processes during the experiments.

References

1. Carr, D.: How Google Works, http://www.baselinemag.com/c/a/Projects-Networks-and-Storage/How-Google-Works-%5B1%5D/
2. HADOOP.COM. Product page, http://www.hadoop.com
3. HBASE.ORG. Product page, http://www.hbase.org
4. Chang, F., Dean, J., Ghemawat, S., et al.: Bigtable: A Distributed Storage System for Structured Data. In: 7th Symposium on Operating Systems Design and Implementation (OSDI 2006), Seattle, WA, USA, November 6-8, pp. 205–218 (2006)
5. Cordes, K.: YouTube scalability Talk (July 14, 2007), http://kylecordes.com/2007/07/12/youtube-scalability/
6. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. In: Proc. of the 19th ACM SOSP, December 2003, pp. 29–43 (2003)
7. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: Proc. of the 6th OSDI, December 2004, pp. 137–150 (2004)
8. Burrows, M.: The Chubby lock service for loosely coupled distributed systems. In: Proc. of the 7th OSDI (November 2006)
9. YAHOO.COM. Product page, http://research.yahoo.com/node/1849
10. APACHE.ORG. Product page, http://lucene.apache.org/nutch
11. Baldeschwieler, E.: Yahoo! Launches world's biggest Hadoop production application (February 19, 2008), http://developer.yahoo.com/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html

12. Borthakur, D.: The Hadoop Distributed File System: Architecture and Design (May 21, 2008), http://hadoop.apache.org/core/docs/r0.17.0/hdfs_design.html
13. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. Communications of the ACM 51(1) (January 2008)
14. Hood, S.: MapReduce at Rackspace (January 23, 2008), http://blog.racklabs.com/?p=66
15. List of companies using Hadoop (June 6, 2008), http://wiki.apache.org/hadoop/PoweredBy
16. Reed, B.: Zookeeper (March 25, 2008), http://research.yahoo.com/node/2120
17. GOOGLE.COM. Product page, http://code.google.com/p/pyshards/wiki/Pyshards
18. Delap, S.: HBase Leads Discuss Hadoop, BigTable and Distributed Databases (April 28, 2008), http://www.infoq.com/news/2008/04/hbase-interview
19. Hoff, T.: How to learn to stop worrying and use lots of disk space to scale (May 21, 2008), http://highscalability.com/how-i-learned-stop-worrying-and-love-using-lot-disk-space-scale
20. FACEBOOK.COM. Product page, http://developers.facebook.com/thrift/
21. HADOOP.CA. Product page, http://www.hadoop.ca
22. Unknown author, Hadoop MapReduce Tutorial (May 21, 2008), http://hadoop.apache.org/core/docs/r0.17.0/mapred_tutorial.html
23. ISO/IEC 9126:1999. Software Engineering – Product quality. Int. Org. for Standardization, ISO 9126 (1999)