Distributed Storage of Large Scale Multidimensional Electroencephalogram Data using Hadoop and HBase

Haimonti Dutta*, Alex Kamil, Manoj Pooleery, Simha Sethumadhavan and John Demme

Abstract. Huge volumes of data are being accumulated from a variety of sources in engineering and scientific disciplines; this has been referred to as the *"Data Avalanche"*. Cloud computing infrastructures (such as Amazon Elastic Compute Cloud (EC2)) are specifically designed to combine high compute performance with high performance network capability to meet the needs of data-intensive science. Reliable, scalable, *distributed* computing is used extensively on the cloud. Apache Hadoop is one such open-source project that provides a distributed file system to create multiple replicas of data blocks and distribute them on compute nodes throughout a cluster to enable reliable and rapid computations. Column-oriented databases built on Hadoop (such as HBase) along with MapReduce programming paradigm allows development of large scale distributed computing applications with ease. In this chapter, benchmarking results on a small in-house Hadoop cluster composed of 29 nodes each with 8-core processors is presented along with a case-study on distributed storage of electroencephalogram (EEG) data. Our results indicate that

Haimonti Dutta
Center for Computational Learning Systems (CCLS), Columbia University, New York 10115.e-mail: haimonti@ccls.columbia.edu

Alex Kamil
School of General Studies, Columbia University, New York, NY 10027.e-mail: alex.kamil@gmail.com

Manoj Pooleery
Center for Computational Learning Systems (CCLS), Columbia University, New York 10115. e-mail: manoj@ccls.columbia.edu

Simha Sethumadhavan
Computer Architecture Laboratory, Department of Computer Science, Columbia University, New York, 10115. e-mail: simha@cs.columbia.edu

John Demme
Department of Computer Science, Columbia University, New York, 10115. e-mail: jdd@cs.columbia.edu

* Corresponding Author

the Hadoop / HBase projects are still in their nascent stages but provide promising performance characteristics with regard to latency and throughput. In future work, we will explore the development of novel machine learning algorithms on this infrastructure.

1 Introduction

The science of the 21st century requires large amounts of computation power, storage capacity and high speed communication. These requirements are increasing at an exponential rate and scientists are demanding much more than is available today. Several astronomy and physical science projects such as CERN's[2] Large Hadron Collider ([27]), Sloan Digital Sky Survey ([33]), The Two Micron All Sky Survey ([1]), bioinformatics projects including the Human Genome Project ([24]), gene and protein archives ([30, 35]), meteorological and environmental surveys ([29, 19]) are already producing peta- and tera- bytes of data which requires to be stored, analyzed, queried and transferred to other sites. To work with collaborators at different geographical locations on peta scale data sets, researchers require communication of the order of Gigabits / sec. Thus computing resources are failing to keep up with the challenges they face.

Traditionally, supercomputers ([32], [23], [18]) are used for highly compute-intensive tasks such as problems involving quantum mechanical physics, weather forecasting, climate research and molecular modeling. They use custom-made CPUs with innovative designs that allow them to perform many tasks in parallel and hence can gain substantial speed over conventional computers. However, the data has to be stored in a separate repository which has to be brought in each time for computation. This is time consuming and limits interactivity; furthermore the programs are written at very low level language and rely on a small number of specific packages written by experts. Also, in a super-computer, users submit jobs in a batch mode; the job is done when resources are available - so it does not support flexible programming and runtime environment.

Cloud computing paradigms are being considered for data intensive science in recent years. The concept of the "cloud" has been envisioned to provide a solution to the increasing data demands and offer a shared, distributed computing infrastructure. The *sharing* of distributed computing resources including software, hardware, data, etc. is an important aspect of cloud computing. Sharing can be dynamic depending on the current need, may not be limited to client server architectures and the same resources can be used in different ways depending on the objective of sharing. Systems such as the Amazon Simple Storage Service ([3]) and Amazon Elastic Compute Cloud ([2]) are good examples of the Storage and Compute Clusters which provide cloud computing infrastructures.

[2] Conseil Europen pour la Recherche Nuclaire - European Organization for Nuclear Research

In this chapter, we present a mechanism for distributed storage of multi-dimensional electroencephalogram (EEG) time series obtained from epilepsy patients on a cloud computing infrastructure (Hadoop cluster) using a column-oriented database (HBase). It is organized as follows: Section 2 presents prior research in storing large time series databases; Section 3 describes preliminaries on Hadoop and Section 4 discusses the basic structure of a column-oriented database HBase. We present a case-study on large-scale storage of intra-cranial EEG data on HBase in section 5. Finally, section 7 concludes the paper and presents directions for future work.

2 Related Work

To the best of our knowledge, there is very little work that addresses the problem of time series data *storage* on large distributed environments. The Harvard Time Series Center[3]) is known to be one of the largest data centers hosting approximately a billion time series mainly from the field of astronomy but also expanding to economics, health and real-estate data.

There is a relatively large body of literature pertaining to analysis and extraction of patterns from time series data ([25, 26]); however, very few are known to scale to large datasets ([31, 12, 34, 36, 28, 13]) stored in distributed environments. Reeves et. al. [31] describe *Cypress*, a framework to archive and query massive time series streams by sparse (frequency and time domain) representations of the data. The sparsity enables archiving of data in a reduced storage space. Trends such as histograms and correlations can be answered directly from the compressed data. Das et. al. [12] consider the problem of distributed eigen monitoring algorithms in petascale astronomy pipelines. They propose an asynchronous algorithm for monitoring principal components of dynamic data streams. Again, the problem of large scale data storage is largely ignored in this work. The Zohmg [4] system is probably closest in spirit to the current chapter in that they describe a large scale data store for aggregated time series data. The goal is to model multidimensional time series as data cubes on top of a distributed, column-oriented database (HBase) to reap the scalability benefits of such databases. In order to import data to Zohmg, the user writes a custom map function and employs Dumbo[4] (a platform that enables usage of streaming Hadoop instances), to execute it on an Apache Hadoop instance. The Zohmg user map function emits triples, which consist of a time-stamp, a hash table of dimensions and their respective values, and a hash table with units and their respective values. A reducer function that is specific to Zohmg is employed to perform aggregation on the output from the map phase. The reducer sums the measurements for each point in an n-dimensional space for each unit. The output from the reducer is interpreted by a custom output reader.

[3] http://timemachine.iic.harvard.edu/publications/

[4] http://www.audioscrobbler.net/development/dumbo/

Since our case-study deals with storage of large-scale EEG data, we discuss related literature in this research area also. A collaboration between the University of Pennsylvania and the Mayo Clinic (which is currently funded by National Institutes of Health) deals with large volumes of intra-cranial electroencephalogram (iEEG) data from epileptic patients. As part of research done in this collaboration, Brinkman et. al. [8] describe a platform for acquisition, compression, encryption, and storage of large-scale EEG data. Continuous, long-term electrophysiological recordings in human subjects undergoing evaluation for epilepsy surgery using intracranial electrodes and clinical macroelectrode arrays generates approximately 3 terabytes of data per day (at 4 bytes per sample). Their work studies real-time data compression techniques enabling random access to data segments of varying sizes. Yet another database that provides invasive EEG recordings for patients suffering from intractable focal epilepsy is hosted at the Epilepsy Center of the University Hospital of Freiburg[5], Germany [6]. Recordings from 21 patients are acquired using 128 channels, 256 Hz sampling rate, and a 16 bit analogue-to-digital converter. A relatively small EEG data set with sampling rate of 173.61 Hz is hosted at the University of Bonn [5].

3 Preliminaries on Hadoop

Apache Hadoop ([20]) inspired by Google Map-Reduce ([14], [15]) and Google File System ([17]), is a framework for supporting data intensive applications on a cluster. It has a free open source MapReduce implementation and was first used on a commercial level for the Nutch ([9]) search engine project. Since then both the industry and academia have been working together to develop tools and architectures for supporting Data Intensive Scalable Computing ([16]) using Hadoop.

MapReduce is a distributed computing framework for large datasets and has two computation phases – `map` and `reduce`. In the `map` phase, a dataset is partitioned into disjoint parts and distributed to workers called `mappers`. The mappers implement compute-intensive tasks (such as clustering) on local data. The power of MapReduce stems from the fact that many map tasks can run in parallel. The output of the `map` phase is of the form $< key, value >$ pairs which are passed to the second phase of MapReduce called the`reduce` phase. The workers in reduce phase (called reducers) then partition, process and sort the $< key, value >$ pairs received from the Map phase according to the *key* value and make final output. For a complex computation task, several MapReduce phase pairs may be involved.
The architecture comprises of two main parts: (1) **Data Storage**, using the Hadoop Distributed File System (HDFS). (2) **Computation**, utilizing MapReduce programming paradigm to meet the computation needs of the clustering algorithm. Figure 1 shows the above components in our cluster. Specifically, these include:

[5] https://epilepsy.uni-freiburg.de/freiburg-seizure-prediction-project/eeg-database

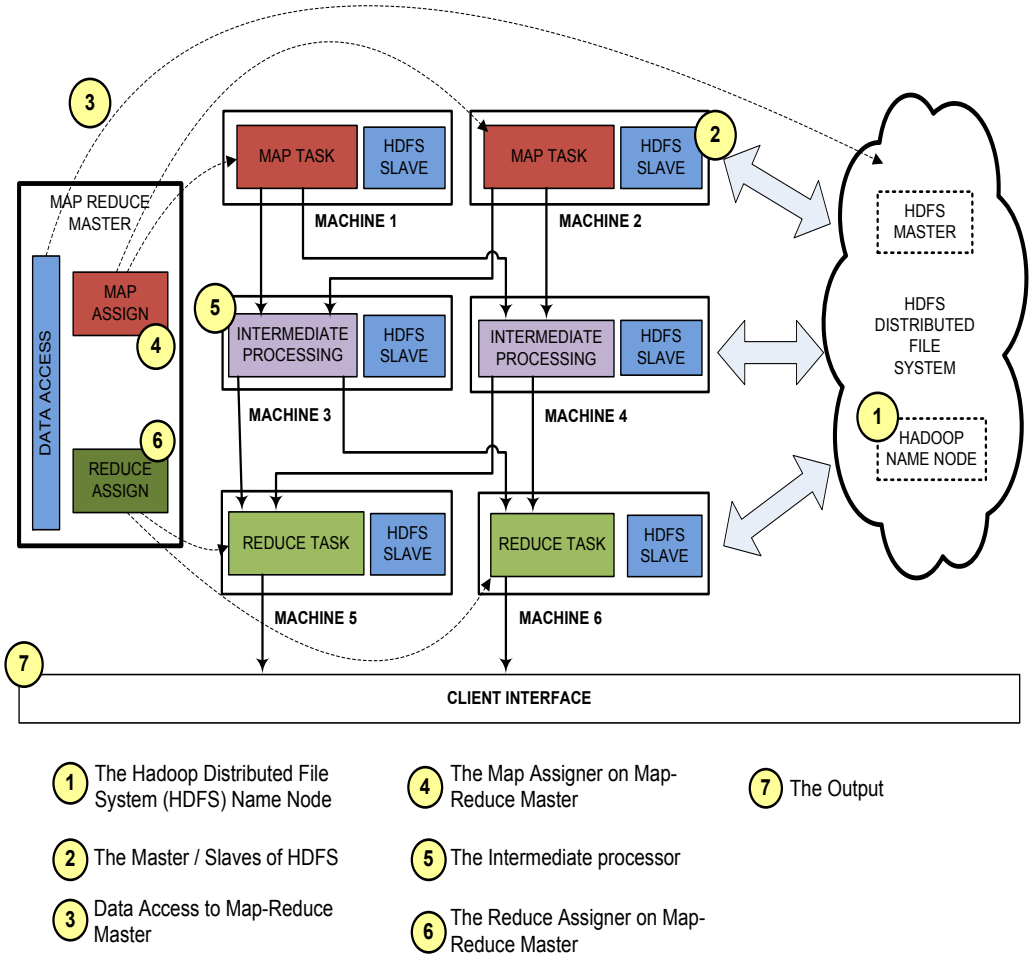

Fig. 1 The architecture of a Cluster illustrating the Map, Reduce and Intermediate operations along with the Hadoop Distributed File System (HDFS).

1. **The Hadoop Distributed File System (HDFS) Namespace**: The Namenode maintains the file system namespace and records any changes made to it. It also keeps track of the number of replicas of a file that should be maintained in the HDFS typically called the replication factor.
2. **The Master / Slaves of HDFS**: A master server manages the file system namespace and regulates access to files by clients. In addition, there are a number of HDFS Slaves, usually one per node, which manage the data associated with that

node. They serve read and write requests from the users and are also responsible for block creation, deletion and replication upon instruction from the NameNode.
3. **Data Access to MapReduce Master**: The HDFS file system will be accessed by a MapReduce (MR) master. The input files to the MR Master can be processed in parallel by different machines in cluster.
4. **The Map Assigner on MapReduce Master**: It stores data structures such as the current state (idle, in-progress or completed) of each map task in the cluster. It is also responsible for pinging the map-workers occasionally. If no response is received from the worker, it assumes that the process has failed and re-schedules the job.
5. **The Intermediate processor**: The intermediate $< key, value >$ pairs produced by map function are buffered in the local memory of machines. This information is sent to the MR Master which then informs the Reduce Assigner.
6. **The Reduce Assigner on MapReduce Master**: This takes in the location of the intermediate files produced from a Map operation and assigns reduce jobs to the respective machines.
7. **The Output**: The output of a Reduce function is appended to a final output file. When all the map and reduce tasks are over, the MR Master wakes up the user program.

Read / Write Benchmarks on a small scale cluster: The in-house cluster available to us for experimentation had 29, 8-core processors each with 24 GB RAM, 1 TB RAID connected via a fiber channel. Apache Hadoop (version 0.20.2) was set-up on this cluster and the IO throughput of the distributed file system was tested. Table 1 summarizes the results and Figures 2 and 3 present the read and write throughputs for varying number of files, keeping the total file size constant at 1000 MB.

Test	No of files	File Size (MB)	HDFS BYTES READ OR WRITTEN	No of Map Tasks	Execution Time (secs)
Write	10	1000	10485760000	19	219.095
Write	30	1000	31457280079	53	403.295
Write	60	1000	62914560082	101	651.326
Write	100	1000	104857600084	150	1013.916
Read	10	1000	10446515887	17	157.752
Read	30	1000	30415428946	51	259.864
Read	60	1000	60844599982	98	387.055
Read	100	1000	100933188734	140	506.269

Table 1 Execution times for Read and Write operations in the Hadoop Distributed File System (HDFS) for a small cluster of 29 nodes.

Having described the MapReduce and Hadoop frameworks, we proceed in the next section to provide a brief review of HBase.

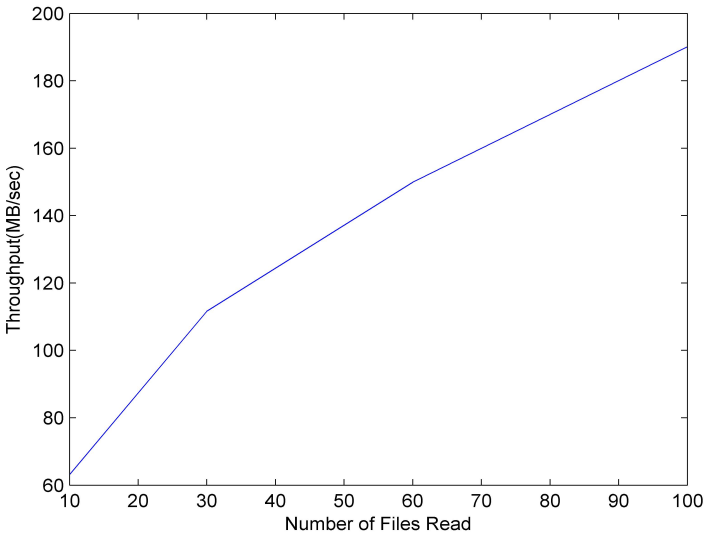

Fig. 2 Throughput vs Number of files Read on the Hadoop cluster (29 nodes).

4 Preliminaries on HBase

HBase ([21], [22]) is an Apache open source project whose goal is to provide Bigtable-like [10] storage (designed to scale to very large databases) for the Hadoop Distributed File System. Applications store rows of data in labeled tables. A data row has a sortable row key and an arbitrary number of columns (illustrated in Table 2). Physically, the table is stored sparsely, so that rows in the same table can have widely varying numbers of columns as illustrated in Table 4, 5 and 7. The empty cells shown in the conceptual view are not stored and hence when queried they will return no value.

The row keys are arbitrary strings of up to 64KB in size and sorted in lexicographical order. Every read or write under a single row key is atomic, which simplifies the handling of concurrent read/writes. Tables are broken up into row ranges called regions (equivalent Bigtable term is `tablet`). Each row range contains rows from start-key to end-key. A set of regions, sorted appropriately, forms an entire table. As a result, reads of short row ranges are efficient and typically require communication with only a small number of machines. Clients can exploit this property by selecting their row keys so that they get good locality for their data accesses.

Column keys are grouped into sets called `column families`, which form the basic unit of access control. A column family must be created before data can be stored under any column key in that family. A column key has the following syntax: <family:qualifier> where <family> and <qualifier> can be byte arrays

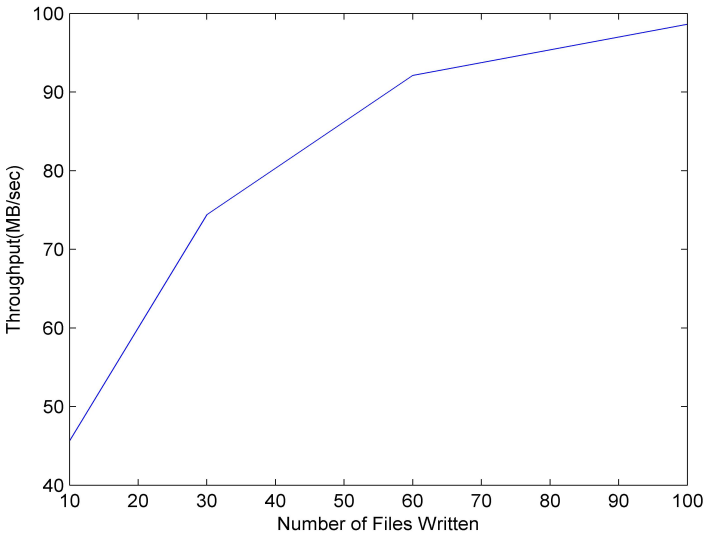

Fig. 3 Throughput vs Number of files Written on the Hadoop cluster (29 nodes).

of arbitrary lengths. HBase stores column families physically close on disk, so that items in a given column family have roughly the same read/write characteristics.

Each cell in a table can contain multiple versions of the same data – these versions are indexed by timestamp. HBase timestamps are 64-bit integers. Applications that need to avoid collisions must generate unique timestamps themselves. Different versions of a cell are stored in decreasing timestamp order, so that the most recent versions can be read first.

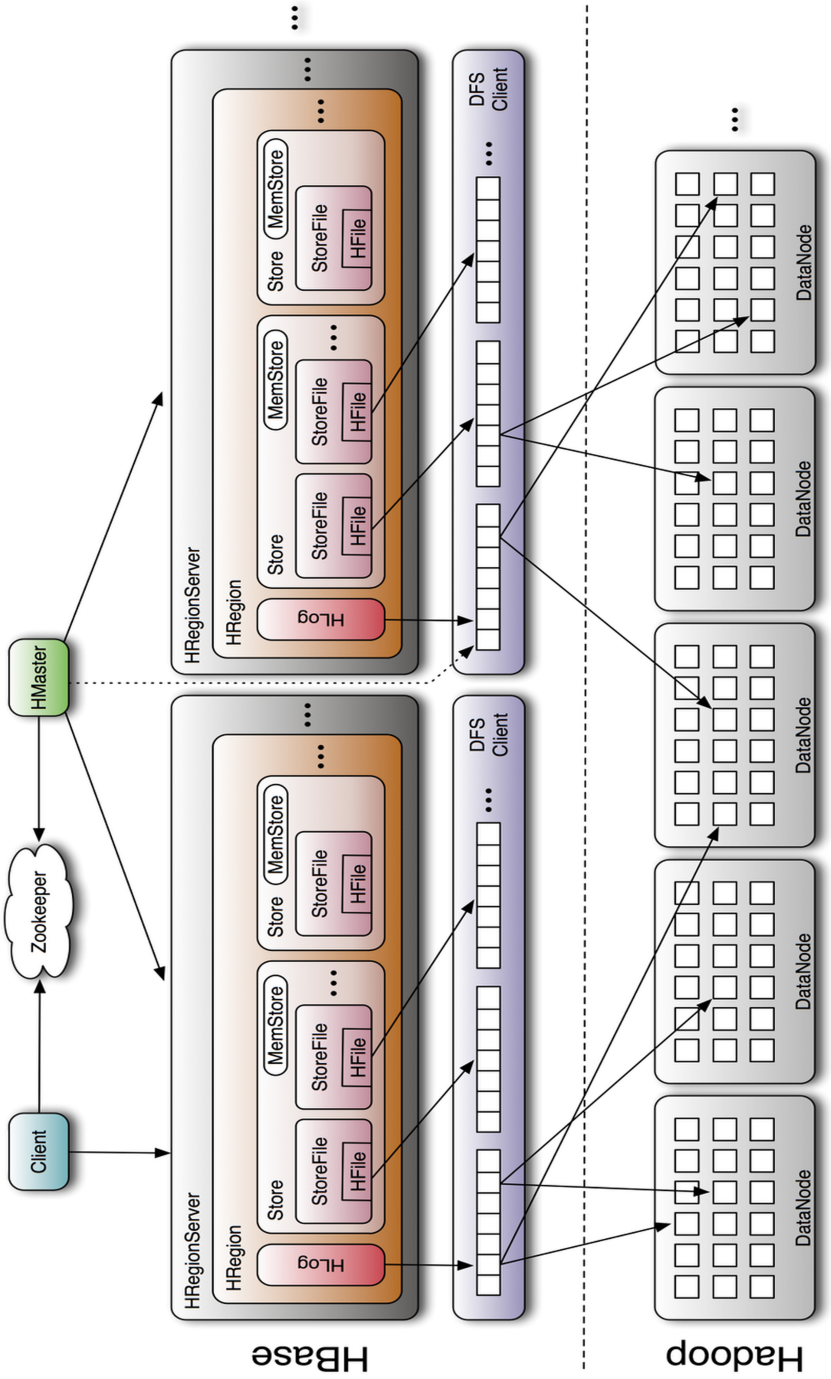

Fig. 4 The HBase architecture

Architecture: The three major components of HBase are

- **The HBase Master:** The HBase Master is responsible for assigning regions to HRegion Servers and monitoring their health. The first region to be assigned is the ROOT[6] region which locates all the META[7] regions to be assigned. Each META region maps a number of user regions which comprise the multiple tables that a particular HBase instance serves. In addition, the HBase Master also handles table administrative functions such as on / off-lining of tables, changes to the table schema (adding and removing column families), etc.
- **The HRegion Server:** The HRegion Server is responsible for handling client read and write requests. It communicates with the HBase Master to get a list of regions to serve and to tell the master that it is alive. When a write request is received, it is first written to a Write-Ahead Log (WAL) called a **HLog** and is stored in an in-memory cache called the Memcache. Reads are handled by first checking the Memcache and if the requested data is not found, the MapFiles are searched for results. The HRegion Server is also responsible for region splits.
- **The HBase Client:** The HBase client is responsible for finding the HRegion Servers that are serving the particular row range of interest. On instantiation, the HBase client communicates with the HBase Master to find the location of the ROOT region. Once located, the client contacts the region server of interest and scans the ROOT region to find the META region that will contain the location of the user region that contains the desired row range.

The Figure 4 provides the architecture[8] of HBase. In the following section, we present a case study on storing multidimensional time series data obtained from epilepsy patients.

Row Key	Time Stamp	Column00	Column01	Column02
Row0	t0	Value=String00_t0	Value=*String*01_t0	Value=String02_t0
Row0	t1	Value=String00_t1		Value=String01_t2
Row0	t3	Value=String00_t3	Value=String01_t3	Value=String02_t3
Row0	t5	Value=String00_t5	Value=String01_t5	

Table 2 Conceptual view of the entire table.

[6] The ROOT table is confined to a single region and maps all the regions in the META table. Each row in the ROOT and META tables is approximately 1KB in size. At the default region size of 256MB, this means that the ROOT region can map 2.6 x 105 META regions, which in turn map a total 6.9 x 1010 user regions, meaning that approximately 1.8 x 1019 (264) bytes of user data.

[7] The META table stores information about every user region in HBase such as start and end row keys, whether region is on or off-line and address that is currently serving the region.

[8] This figure is obtained from http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

Row Key	Time Stamp	Column00
Row0	t0	Value=String00_t0
Row0	t1	Value=String00_t1
Row0	t3	Value=String00_t3
Row0	t5	Value=String00_t5

Table 3 Physical storage of column00 for Table 2.

Row Key	Time Stamp	Column01
Row0	t0	Value=String00_t0
Row0	t3	Value=String00_t3
Row0	t5	Value=String00_t5

Table 4 Physical storage of column01 for Table 2.

Row Key	Time Stamp	Column02
Row0	t0	Value=String00_t0
Row0	t3	Value=String00_t3

Table 5 Physical storage of column02 for Table 2.

5 Case Study: Large Scale Storage and Indexing of EEG data

A large scale EEG database obtained from the University Hospital of Freiburg Germany was used for benchmarking experiments and testing the design of the multi-dimensional time series index using Apache Hadoop and HBase.

5.1 Data set description

Epilepsy is a chronic neurological disorder characterized by recurrent, unprovoked seizures that manifest in a variety of ways, including emotional or behavioral disturbances, convulsive movements, and loss of awareness. EEG has been used in the evaluation of epilepsy and related brain disorders since the 1930s ([7]). Interpretation of EEG has traditionally been carried out by visually scanning for recognizable patterns.

Row Key	Time	Channel 1	Channel 2	$\cdots$	Channel 128
R0	200602190201237000	value:[123]	value:[345]	$\cdots$	value:[0]
R0	200602190201237001	value:[876]	value:[123]	$\cdots$	value:[123]
R0	200602190201237002	value:[56]	value:[348]	$\cdots$	value:[121]

Table 6 The conceptual table storing the iEEG Data

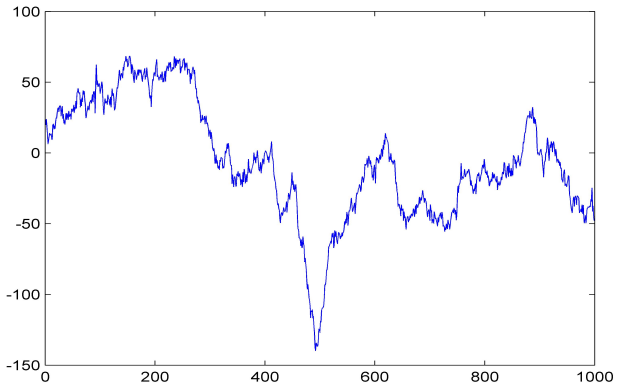

Fig. 5 A snippet of EEG time series data. The X-axis plots the time (in seconds) and the Y-axis plots the amplitude of the signal.

The EEG database we use for benchmarking and experimentation contains invasive EEG recordings of 21 patients suffering from medically intractable focal epilepsy. The data was recorded during an invasive pre-surgical epilepsy monitoring at the Epilepsy Center of the University Hospital of Freiburg, Germany[9] and was recorded using a Neurofile NT digital video EEG system with 128 channels, 256 Hz sampling rate, and a 16 bit analogue-to-digital converter. No prior signal processing (such as application of notch or band pass filters) has been done on the data. For each of the patients, there are datasets called "ictal" and "interictal", the former containing files with epileptic seizures and at least 50 min pre-ictal data and the latter containing approximately 24 hours of EEG-recordings without seizure activity. Figure 5 shows a snippet of EEG time series data obtained from a patient suffering from epilepsy. For each point in time, the amplitude of the signal is recorded. Thus the conceptual view of the EEG data can be thought of as shown in Table 6. For each time point, there are recordings for all the 128 channels. This representation corresponds to a row-oriented database. Using a column-oriented representation, the data can be stored as shown in Table 7.

Row Key	Time Stamp	Channel1
Row0	20060219020123700	123
Row0	200602190201237001	876
Row0	200602190201237002	56

Table 7 Physical storage of Channel1 for Table 6.

[9] http://www.uniklinik-freiburg.de/epilepsie/live/index_en.html

6 Benchmarking Workloads using the Yahoo! Cloud Serving Benchmark

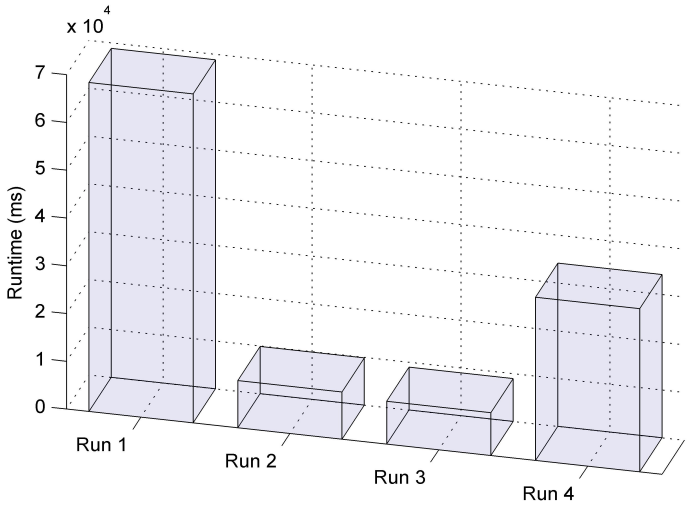

Fig. 6 Runtime over 4 different runs of workload comprising of 50% read and write operations on 10,000 records (6 nodes).

The Yahoo! Cloud Serving Benchmark (YCSB) [11] helps to create a standard benchmarking framework to assist in the development of cloud computing systems. The YCSB client is a java program for generating data to be loaded to the database and operations to be made on the workload. The workload executor manages multiple threads; each thread executes a sequential series of operations by which load, read and write commands are executed on the workload. We use YCSB to test the performance of the EEG dataset stored in HBase running on a Hadoop Distributed File System in a small in-house cluster using different workload compositions (read and write access to the database). The Hadoop / HBase combination allows us to scale horizontally by simply adding more nodes; this framework also helps to deal with fault tolerance, data partitioning and provide an elaborate API so that the client code can be implemented on a high level of abstraction without the users worrying about low level details such as distributed file system implementation, scheduling and task coordination. HBase allows random access reads and writes with minimum disk IO overhead. Our in-house cluster is configured in a student laboratory and there were frequent workstation restarts and shutdowns which resulted in loss

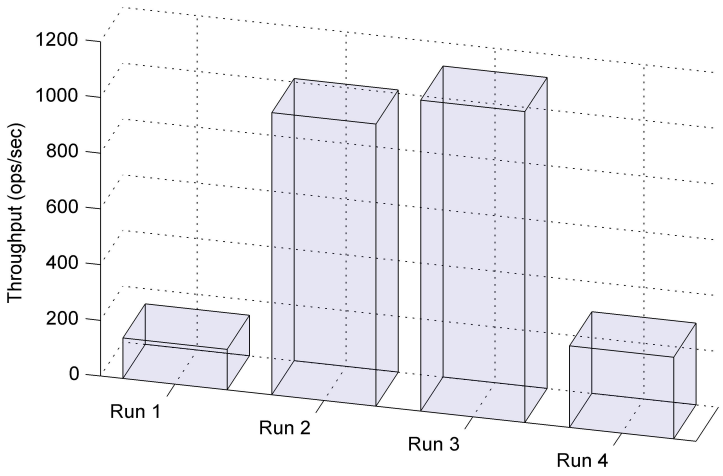

Fig. 7 Throughput over 4 different runs of workload comprising of 50% read and write operations on 10,000 records (6 nodes).

of nodes, but Hadoop (HDFS) was able to recover since the data was replicated with replication factor of 3. Thus, there were no outages in HDFS availability.

To LOAD 1 million records of EEG data into the column-oriented database HBase, the throughput was found to be 1511.46 operations / sec with an average latency of 0.6469 ms. The YCSB client allows to benchmark new database systems by implementing the read, insert, update, delete or scan methods to represent the standard "CRUD" operations (Create, Read, Update and Delete operations). The workload executor is created and shared amongst worker nodes. The first workload we tested consisted of 50% read and 50% write operations on 10,000 records in the database of size 1 million loaded as discussed above. The figures 6, 7 and 8 illustrates the statistics obtained over 4 different runs of the same workload. It is important to note that our usage of HBase and Hadoop will be mostly write once, read many times so the read latency and throughput are more important than write latency and throughput. Read latency was found to be less than a milli-second on average under controlled load which satisfies our initial performance criteria.

The next experiment demonstrated the effect of the number of threads on the runtime, throughput and average latency of the read and update commands. Table 8 demonstrates the results. If the maximum number of threads is set at 25, for 1 Million records, the throughput achievable is approximately $7,000$ operations / sec. The throughput generally increases with number of threads used for experimentation; however, this is limited by the number of database accesses made. For example, empirical results revealed that if the number of threads is increased from 25

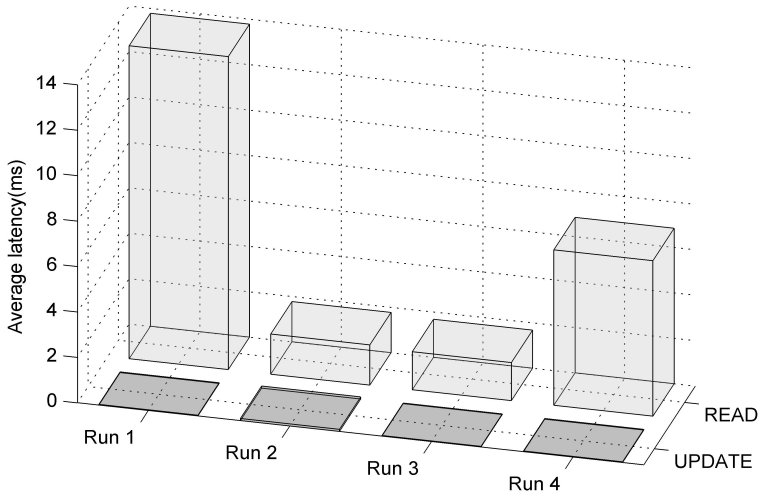

Fig. 8 Average Latency over 4 different runs of workload comprising of 50% read and write operations on 10,000 records(6 nodes).

No of Threads	Runtime (ms)	Throughput (ops / sec)	Avg READ Latency (ms)	Avg UPDATE Latency (ms)
10	200310	49.92	7.93	0.372
30	3071	3256.26	3.11	2.46
50	2690	3717.47	5.09	4.28
100	2466	4055.15	11.46	11.17
150	2217	4510.6	19.24	19.81
200	2622	3813.88	33.47	28.27

Table 8 Average Latency with varying number of HBase client threads

to 100, performance eventually drops as there is a trade-off between the number of database accesses made and parallelization of jobs run[10]. This indirectly establishes a limit on how many clients the system can serve concurrently. It must also be noted that there are some inconsistencies across runs in terms of throughput since HBase at the time of the experiments was under active development and relatively unstable. Its performance and stability have been significantly improved in subsequent versions[11]. Finally, the capacity of our system is limited by the hardware and networking configuration of the cluster used. In order to avoid this dependency, we plan

[10] We ascertained that this is not a limitation on the client and it was not overloaded in terms of either cpu or bandwidth utilization

[11] HBase 0.20.6 and the more recent development releases which specifically addressed and fixed many bugs which affected performance and stability

to experiment in future with Amazon AWS (`http://aws.amazon.com/`) as a cloud hosting platform where the inter-cluster network bandwidth is much higher (1-Gbps Ethernet on Amazon EC2 vs 100Mbps in our stand-alone setup) than the cluster setup we used for experiments here – large clusters such as Amazon's EC2 also provide more powerful server grade nodes which provide more compute power than workstations typically used in small clusters.

7 Conclusion and Future Work

In many scientific domains such as astronomy, social science and medicine researchers are faced with a data avalanche. Cloud computing paradigms are being used in these domains for data-intensive science. A popular distributed file system that has been used for large-scale data storage is the Apache Hadoop framework. Column-oriented databases built on the Hadoop, such as HBase, are known to have several advantages over traditional row-oriented databases. MapReduce enables development of large-scale computational tasks on the Hadoop framework. In this chapter, we first provide a brief overview of the Hadoop and HBase infrastructures and then present benchmarking results on a small in-house cluster composed of 29 nodes with 8-core processors each. We present a case-study on distributed storage of multi-dimensional EEG data using Hadoop and HBase and present extensive scalability results using the Yahoo! Cloud Serving Benchmark (YCSB). Our results indicate that the Hadoop and HBase ecosystem including the dependencies (services like zookeeper) are still quite immature in terms of stability but promising in terms of the performance characteristics with regard to latency and throughput. The issues pertaining to stability of Hadoop and HBase are being investigated by the project developers in more recent releases. Future work involves design and benchmarking of machine learning algorithms on this infrastructure and pattern matching from large scale EEG data.

Acknowledgements Funding for this work is provided by National Science Foundation award, IIS–0916186. Data for this project was provided by the Freiburg Seizure Prediction EEG Database (FSPEEG). The authors would like to thank administrators of the CLIC Lab Cluster at the Department of Computer Science, Columbia University for help with cluster set-up and experimentation; Dr. David Waltz, Dr. Catherine A Schevon, Dr. Ronald Emerson, Phil Gross and Shen Wang provided their insightful comments during different phases of the project.

References

1. 2-Micron All Sky Survey. http://pegasus.phast.umass.edu, Website.
2. http://aws.amazon.com/ec2/, Website.
3. http://aws.amazon.com/s3/, Website.

4. Per Andersson and Fredrik Mollerstrand. Zohmg-a large scae data store for aggregated time-series-based data. Master's thesis, Chalmers University of Technology, 2009.
5. R. G. Andrzejak, K. Lehnertz, C. Rieke, F. Mormann, P. David, and C. E. Elger. "Indications of nonlinear deterministic and finite dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state". *Phys. Rev E*, (64, 061907), 2001.
6. R. Aschenbrenner-Scheibe, T. Maiwald, M. Winterhalder, H.U. Voss, J. Timmer, and A. Schulze-Bonhage. How well can epileptic seizures be predicted, an evaluation of a nonlinear method. *Brain*, 126:2616–2626, 2003.
7. H. Berger. Uber das elektroencephalogramm des menschen (on the electroencephalogram of man). *Archiv fiir Psychiatrie und Nervenkrankheiten*, 87:527–570, 1929.
8. Benjamin H. Brinkmann, Mark R. Bower, Keith A. Stengel, Gregory A. Worrell, and Matt Stead. Large-scale electrophysiology: Acquisition, compression, encryption, and storage of big data. *Journal of Neuroscience Methods*, 180(1):185 – 192, 2009.
9. M. Cafarella and D. Cutting. Building nutch: Open source search. In *ACM Queue*, April 2004.
10. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):1–26, June 2008.
11. B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. *ACM Symposium on Cloud Computing, ACM, Indianapolis, IN, USA*, 2010.
12. K. Das, K. Bhaduri, S. Arora, W. Griffin, K. Borne, C. Giannella, and H. Kargupta. Scalable Distributed Change Detection from Astronomy Data Streams using Local, Asynchronous Eigen Monitoring Algorithms. In *In Proc. of the SIAM International Conference on Data Mining*, Sparks, Nevada, 2009.
13. R. Dave. *"Scaling Astronomy"*. Oreilly Ignite 4, 2008.
14. J Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 2004 OSDI Conference*, 2004.
15. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.
16. http://www.cs.cmu.edu/ bryant/, Website.
17. S. Ghemawat, H. Gobioff, and S. T. Leung. The google file system. In *The 19th ACM Symposium on Operating Systems Principles*, lake George, NY, 2003.
18. http://grape.mtk.nao.ac.jp/grape/news/ABC/ABC-cuttingedge000602.html, Website.
19. S.J. Graves, H. Conover, K. Keiser, R. Ramachandran, S. Redman, J. Rushing, and S. Tanner. "Mining and Modeling in the Linked Environments for Atmospheric Discovery (LEAD)". In *Huntsville Simulation Conference*, Huntsville, AL, Oct. 19, 2004 2004.
20. http://hadoop.apache.org/core/, Website.
21. http://hbase.apache.org/, Website.
22. http://wiki.apache.org/hadoop/Hbase/HbaseArchitecture, Website.
23. F. H. Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002.
24. The human genome project. http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml, Website.
25. Eamonn J. Keogh. Recent advances in mining time series data. In *PKDD*, page 6, 2005.
26. Eamonn J. Keogh. A decade of progress in indexing and mining large time series databases. In *VLDB*, page 1268, 2006.
27. http://lhc.web.cern.ch/lhc/, Website.
28. J. Lin, M. Vlachos, E. Keogh, and D. Gunopulos. Iterative incremental clustering of time series. *In proceedings of the IX Conference on Extending Database Technology.*, 2004.
29. World data center for meterology. http://www.ncdc.noaa.gov/oa/wmo/wdcamet.html, Website.
30. The protein data bank (pdb). http://www.rcsb.org/pdb/Welcome.do, Website.
31. Galen Reeves, Jie Liu, Suman Nath, and Feng Zhao. Managing Massive Time Series Streams with Multi-Scale Compressed Trickles. In *In Proc. of the 35th Conference on Very Large Data Bases*, Lyon, France, 2009.

32. http://www.sdsc.edu/, Website.
33. Sloan Digital Sky Survey. http://www.sdss.org, Website.
34. Jin Shieh and Eamonn J. Keogh. *i*sax: indexing and mining terabyte sized time series. In *KDD*, pages 623–631, 2008.
35. The swiss-prot protein knowledge base. http://www.expasy.org/sprot/, Website.
36. Dragomir Yankov, Eamonn J. Keogh, and Umaa Rebbapragada. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. In *ICDM*, pages 381–390, 2007.