Storage for Heterogeneous Data Stream

Rohan Sunil Karwa
University of California
Los Angeles
rskarwa@cs.ucla.edu

Amit Karyekar
University of California
Los Angeles
amit.karyekar@cs.ucla.edu

Drumil Jaswani
University of California
Los Angeles
drumiljaswani@cs.ucla.edu

Rohit Joshi
University of California
Los Angeles
rohitjoshi@cs.ucla.edu

ABSTRACT

Today, we have numerous systems/devices that generates data stream. All today's scalable system for storage of large amount of data were not designed for storage of heterogeneous data streams. Usage of today's storage system for the stream persistence and data query would be highly inefficient. Hence, in this project, we aimed at designing system for storage of heterogeneous stream data along with support of efficient data retrieval. The system design can scale to store limitless data, and can be ran on commodity machines. The designed system supports range queries as a first class operation. The system has inbuilt support for pattern finding and thus supporting pattern mining. Also, by adding synchronization to the heterogeneous data streams, we are able to identify patterns in each data stream more accurately. In addition, it enables the system to successfully capture any dependencies that may exist among the heterogeneous data streams. Designing such system is first step towards building the system, and our work open up large number of avenues for future work.

Keywords

Data Streams, Distributed Storage System, Heterogeneous Data Stream, SAX

1. INTRODUCTION

In recent years, with the advancement of the technology it is now possible to create small devices having small computing power in order to generate the continuous data and propagate the generated data to server for persistence. Apart from such devices, today it is not uncommon in the areas like financial applications (stock monitoring), network monitoring (packet monitoring), security, telecommunications data management, web applications, manufacturing, sensor networks [1], to generate infinite data stream, and have requirement of persisting the generated data.

Today, though the storage system has become cheaper, main challenge is to develop a system that can scale as per the data generation and collection rate. To add to the problem, today also it is not uncommon to have requirements for synchronization and storage of heterogeneous data stream. In simple terms, heterogeneous data stream can be thought as collection of multiple homogeneous data stream generated from some common phenomena. Example of heterogeneous data stream could be the patient health monitoring system that generates different continuous data streams like Heart Rate, Blood Pressure, ECG, etc. Another example of heterogeneous data stream could be group of sensors monitoring Temperature, Pressure, and Volume of some chemical system.

Key challenges associated with designing a scalable system for the storage of the data stream is the infinite nature of the incoming stream data. Also, the designed system needs to support efficient data queries alognwith pattern queries on the persisted data. System should have smart mechanism for summarizing the data, handling input data frequency rate, arrival of old data, supporting compaction for old data, and importantly should be resilient to machine failures.

Though today we have scalable systems for storage of large amount of data, using the same system for storing heterogeneous data stream would not work well. Primary reason for this is that current systems are not designed to support range queries as a first class operation and most of them [3], [8] assumes data model as key/value pairs which would make support of range queries as highly inefficient. Also, as per our knowledge, there doesn't exists any system that has inherent support for pattern finding on the persisted data as per the chronological order.

Aim of our project was to design a system for efficient and scalable data storage for heterogeneous data stream. The system should support efficient time range queries for data retrieval. Also, the data model of the system needs to be such that it easily facilitates pattern-finding queries on the persisted data. Also in the design, the challenges mentioned above should be considered and a robust design is to be developed.

Novelty of our work and key contributions:

- System design for the storage of heterogeneous data stream. Proposed system can potentially scale to limitless data, and run on thousands of commodity machines.
- Proposed system handles varied incoming data frequency rate, arrival of old data, support compaction for old data, and is resilient to machine failure.
- Apart from data persistence, system design also sup-

port efficient data range queries for retrieving stored data. Also, the system supports pattern queries to identify the time range when the pattern occurred. This inbuilt support circumvents any requirement for writing any wrapper system for pattern mining.

- In order to minimize the loss of information because of approximation of data stream via window approximation, an unique data model with the synchronization approach is proposed. We introduced notion of *model* for on the fly learning and prediction of the pane length and fluctuation. This is achieved by keeping the pane length adaptable as per the fluctuations in the heterogeneous data stream. System governs the data summarization and approximation as per the predicted value by the *model* of data stream having maximum fluctuations.

Report is organized in following way. In section 2, we mention related work. In section 3 and section 4 we present our Data Model and System Design respectively. SAX implementation and its associated evaluations are mentioned in section 5 and section 6 respectively. In section 7, we describe advanced data model for achieving fine grained synchronization using adaptive pane length. In section 8, we mention about limitation of our system design and future work for the project, and we conclude in section 9.

2. RELATED WORK

With best of our knowledge, no dedicated system exists for storage of heterogeneous data streams. In this section, we describe in brief about the scalable storage systems for persisting large amount of data. As mentioned in the Introduction section of the report, usage of such systems directly for the storage and retrieval of the stream data would be highly inefficient.

2.1 Google Big Table

Google Big Table [3] is a distributed storage system specially designed for storage of petabytes of data across thousands of commodity servers. Big Table stores all its data in the key/value fashion using SSTable data structure. It support Read/Write operation on each row and all operations at row level are atomic. Row ranges are partitioned into tablets, and Tablet Servers are designed for managing and serving data from tablets. Big Tables uses Distributed Lock Service called as Chubby [4]. Chubby ensures that at any given point of time there is atmost one active master. Chubby helps to bootstrap location of Big Table data, discover Tablet Servers, stores Big Table schema information and provides Access Control.

2.2 Windows Azure Storage

Windows Azure Storage (WAS) [5] is also a cloud storage systems which provides its client service of storage of limitless data for any time duration. WAS support different data models (Queues, Blobs, Tables) for data storage. All the stored data is locally and geographically replicated. WAS provides strong consistency for the stored data. All the data stored in the WAS is stored in Extent Nodes. WAS has a nice concept of *Sealed* and *Unsealed* Extent Nodes. When the Extend Node reaches its storage capacity, such extend nodes are marked by the system as *Sealed*. Once Extent Nodes gets *Sealed*, they then only support Read operation. Extend Nodes, which are still not full are called as *Unsealed* and they support both Read/Write operation. *Sealing* of Extent Nodes, significantly improves the read performance as all the operations on the Node data can be done by without acquiring locks, as the data is immutable.

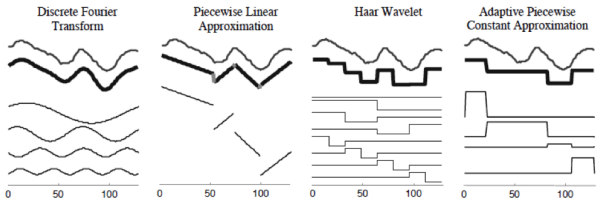

Figure 1: The most common representations for time series data mining. Each can be visualized as an attempt to approximate the signal with a linear combination of basis functions.

3. DATA MODEL

3.1 Time Series Representations

Time series data mining has been an area of focus in the past decade. The following review is a brief introduction, with [2] providing an in depth review.

3.1.1 Time Series Data Mining Tasks

Primarily the research interest in time series data mining has been in the following areas

Indexing: Given a query (time series) Q, and some similarity/ dissimilarity measure D (Q, C), find the most similar time series in the database DB [2]

Clustering: Find the natural groupings of the time series in database under some dissimilarity measure D (Q, C)

Classification: Given an unlabeled time series Q, assign it to one of more predefined classes.

Summarization: Given a time series Q with n data points where n is very large, perform approximation of Q via retaining features.

Anomaly Detection: Given time series Q and model of normal behavior find all the sections of Q that contain anomaly.

3.1.2 Most Common Time Series Representations

A suitable choice of representation greatly affects the ease and efficiency of time series data mining. Considering these lines, many time series representations have been introduced. Most common of all have been illustrated in Figure 1.

A crucial observation is that the above representations are real valued. This limits the algorithms, data structures and definitions for them. To exemplify, in anomaly detection we cannot meaningfully define the probability of observing any particular set of wavelet coefficients, since the probability of observing any real number is zero. Such limitations have

C	A time series $C_1, C_2, C_3 \ldots C_n$
$\bar{C}$	A piecewise aggregated approximation of time series. $\bar{C} = \overline{C_1}, \overline{C_2}, \overline{C_3}, \ldots \overline{C_w}$
$\hat{C}$	A symbol representation of time series $\hat{C} = \widehat{C_1}, \widehat{C_2}, \widehat{C_3}, \ldots \widehat{C_w}$
w	The number of PAA segments representing the time series
a	The alphabet size (e.g. for set of alphabets {a, b, c} alphabet size is 3)

Figure 2: Notations of SAX.

lead the researchers to consider symbolic representation of time series.

3.2 Symbolic Time Series Representation

Researchers have believed that representing the real valued time series in the form of symbols would be useful in the field of text processing, bioinformatics apart from former "batch only" problems.

Traditional systems for symbolic representation of time series have three limitations. To start with, dimensionality of symbolic data is the same as original data and the data mining algorithms scale poorly with dimensionality. Secondly, the distance measures defined on symbolic approaches have little correlation with the distance measures defined on original time series. Lastly, most of the symbolic approaches need to have an access to all the data before creating the symbols which defies its application in streaming algorithms.

In the paper [2], a novel symbolic representation has been introduced with a support for dimensionality reduction, distance measures to be defined on the symbolic approach that lower bound the distance measure on the original series. It concludes to execute the algorithm with infinitesimal time and space overhead.

3.3 Symbolic Aggregate Approximation (SAX)

3.3.1 Introduction

SAX enables time series consisting of N data points to be reduced to a sequence of symbols of length w (where w<n and typically w $<<$ n). The Figure 2 illustrates the notations used to represent SAX.

SAX can be visualized as a two-step process. At first, we transform the time series to piecewise aggregated approximation. Then, symbolize the PAA representation into a discrete string.

3.3.2 Benefits of SAX

Dimensionality Reduction: The length of time series vector can be compressed using well defined procedure of implementing PAA.

Lower Bounding: The distance measure between two symbolic strings lower bounds the actual distance between those two time series. However, the observation that the symbolic distance measure lower bounds PAA distance measure is seemingly more interesting. Since we can prove our desired result by transitivity.

3.3.3 Converting Time Series to SAX Representation

Step 1: Piecewise Aggregated Approximation (PAA) - Time series can be represented in a w dimensional space by a

$$\bar{C} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} C_j$$

Figure 3: Formula for calculating ith element of PAA vector space.

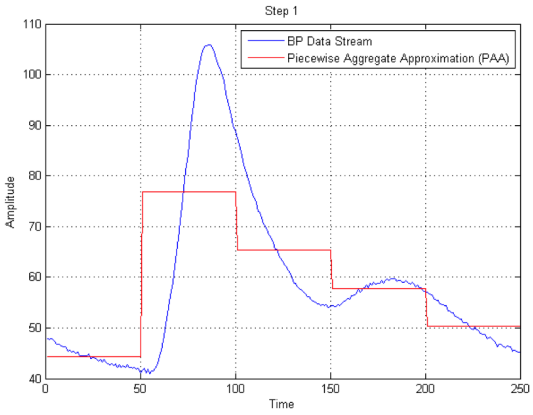

Figure 4: PAA to reduce 250 data points (1 window) to 5 dimensions (5 window panes).

vector whose ith element can be computed using formula shown in figure 3.

The dimensionality reduction is achieved by dividing the time series of n dimensions into w window panes and computing the mean of points corresponding to these window panes. The representation can be thought of as attempt to approximate the original time series with a linear combination of box functions as shown in Figure

To exemplify, given MIT Dataset of four Data Streams (EEG, BP, ECG, Respiration), consider the Figure 4. A key observation in due course is before computing PAA, we normalize the time series. Since we will not be able to compare two time series with different amplitudes and offsets.

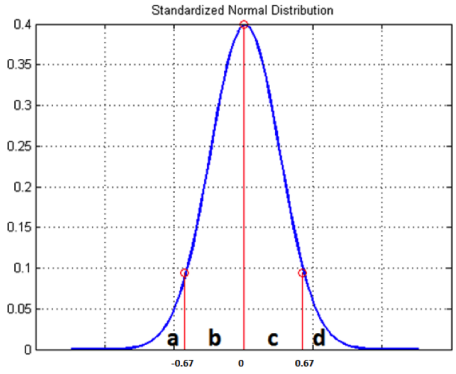

Figure 5: Standard Normal Distribution.

β_i \ a	3	4	5	6	7	8	9	10
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4			0.84	0.43	0.18	0	-0.14	-0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28

Figure 6: Standard Normal Distribution Table.

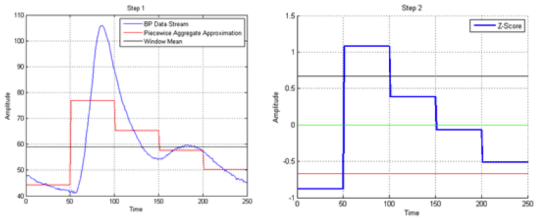

Figure 7: Z-Score Computation.

Step 2: Discretization - Having computed the PAA of time series, discretization can be achieved by assigning symbols, to each window pane, with equal probability. This can be realized using standardized normal distribution, as proved in [2] and shown in Figure 5, since normalized time series follows Gaussian distribution.

We can define the breakpoints that will help us define k-equal sized areas under Gaussian distribution. Breakpoints are a sorted list of Numbers from Ba ... Ba-1 such that the area under a N (0, 1) Gaussian curve from Bi to Bi+1 = 1/a (B0 and Ba are defined as -infinite and +infinite, respectively) as shown in Figure 6.

Once we obtain the breakpoints depending on the number of symbols we want to use, we can employ two steps to discretize the PAA. Firstly, we compute the z-score corresponding to each PAA. Secondly we map the z-score to symbol. Here, Z-score is computed as (PAA Value - Mean of Window)/(Standard Deviation of Window).

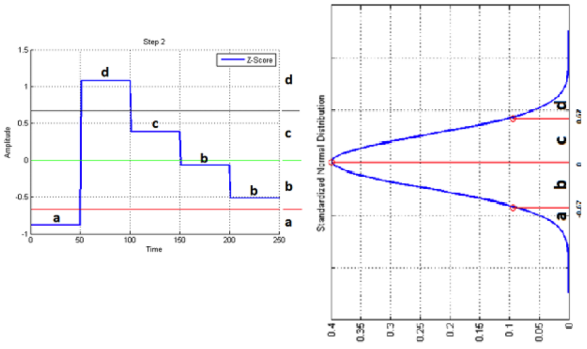

Figure 8: Symbol Assignment.

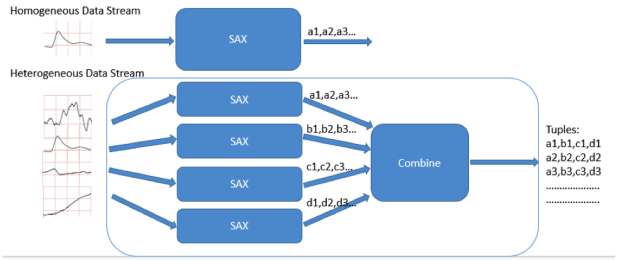

Figure 9: SAX as a black box.

To exemplify consider Figure 7 and Figure 8.

3.4 Usage of SAX for modeling Heterogeneous Stream Data

Having created the sequence of symbols (using piecewise aggregation and symbol mapping through Gaussian distribution), we can associate a homogeneous data stream as an input to the SAX system. On the same lines, incorporating heterogeneous data stream can be visualized to be intuitive as depicted in Figure 9.

To exemplify, consider four data streams forming a heterogeneous data stream. The data points corresponding to these data streams are segregated, on a per window basis, to form an input to corresponding SAX models. These models compute the symbolic sequence, on a per window-pane basis, in accordance with the algorithmic steps. Finally, these symbolic sequences are represented in tuple format to enable persistent storage. To conclude, the simplicity of operation lies in the perception of heterogeneous data stream to be a collection of homogeneous data streams.

Along with the key benefits of SAX being assimilated, our data model can be sought as a black-box i.e. an independent module of system design which is flexible enough to configure/tailor with respect to requirements.

4. SYSTEM DESIGN

As mentioned in the Related Work and Introduction section, as far as our knowledge, there does not exists any dedicated system for the storage of heterogeneous data streams and efficiently retrieving stored data and supporting pattern queries on the persisted data. As part of the survey on popular scalable storage systems, we extensively studied most popular storage systems [3], [5], [9], [8]. As part of survey, we found that we need not completely design new system from scratch, but instead we can use Big Table's system design to start and alter its design to fit to the storage and querying requirements of the heterogeneous data streams. We decided to chose Big Table's design over other system's design because we believed that Big Table's design was close to what we required for persistence of heterogeneous data streams. Also, we borrowed some design principles from Windows Azure Storage [5], like *Sealed* and *Unsealed* Extent Nodes, in order to improve the read performance. Apart from existing components of the Big Table, we introduced additional new components in order to facilitate our storage use case. In this section, we would describe our system design, and emphasize on the components which are modified

and are different than the components in the traditional Big Table.

4.1 Key Design Consideration

4.1.1 Time Series Data is Immutable

One of the key characteristics of the data stream data is that the data is immutable. This provided us to make lot of significant design decision inorder to improve upon the performance of the system.

4.1.2 Chronological Order of the Input Data Stream

One of the fundamental queries expected on the persisted stream data are range queries (where we are interested to get the data between some time range). As the arrival of the stream data is in chronological order, we can take massive advantage of this fact in designing the system in order to increase the performance for the range queries.

4.1.3 Support for parallel query processing

One of the crucial things considered while designing the system was to have data retrieval as fast as possible from the system. Thus special attention was given to the design in order to support parallel query processing for retrieving the data.

4.2 Building Blocks

All important system components are shown in figure 10. Below is brief description about different system components

4.2.1 BSTStore (Binary Search Tree Store)

Google Big Table stores all its data in the SSTable, which is nothing but key/value store. Key/Value store provides fast lookup operation when we exactly know the key whose corresponding value we are interested in. But, in our use case we are more interested for doing range queries and so key/value data structure is not right fit for us. In order to support faster data retrieval for range queries we decided to use Binary Search Tree data structure for storage of data instead of key/value store. So, in the Big Table's design, we replaced SSTable with our new component BSTStore for storing the data which enables faster retrieval of data in case of range queries. Just like SSTable, BSTStore contains sequence of blocks and each block stores the data in the Binary Search Tree for a time range. Block size is predetermined and is same as that of Big Table's block size of 64KB. Each BSTStore contains a Block Index, which enables to locate the block's location. Each time BSTStore is opened, Block Index is loaded into memory for doing the range lookup.

Size of BSTStore is same as that of underneath GFS chunk. This size synchronization enables to achieve data locality and fast data retrieval. In GFS, chunk is fundamental replication unit, and hence we can be sure of the fact that all data present in the BSTStore would be stored on one replica, and fetching of data from a BSTStore would be fast as all the data resides on one machine (one chunk) ensuring data locality.

4.2.2 Tablet

Notion of Tablet in our design is similar to that of Tablet in Big Table. From system design perspective, Tablet is the fundamental unit for managing all the data. For system, it doesn't matter how Tablet underneath stores the data. In our design each tablet manages data for a specific continuous time range via holding reference to multiple BSTStores. Tablet maintains a table like structure containing time range and corresponding BSTStore which contains the data for the time range. Tablet can also be thought as an abstraction storing list of BSTStores.

4.2.3 Tablet Server

Notion of Tablet Server in our design is similar to that of Tablet Server in Big Table but with few differences.

In Big Table, Tablet Server uses the notion of Memtable for storing all the mutation information in memory in case of updates/writes. Data in the memtable is pushed/persisted asynchronously to underneath tablet. This approach allows Big Table to achieve high write throughput. As the memtable's data is stored in memory of Tablet Server, Tablet Server logs all its action on the memtable in commit logs which are persisted in GFS. In case of Tablet Server failure, these log files are read and the memtable is reconstructed. In case of read operation, Tablet Server fetches the corresponding data from Tablet (from GFS) and also checks memtable entry for corresponding read request, apply the required transformation and replies back with the data. This way, Tablet Server manages its read and write operation.

Though this is a great design optimizing both read and write throughput, we believe that we can simplify this design in case of storage of data stream data, primarily because the data is immutable. Memtable design is useful when the data is mutable. Thus, we can relax Tablet Server's responsibility of maintaining memtable, commit/redo logs, and complex read operations. We do not need memtable, and each Tablet Server can only be responsible for serving read request.

Tablet Server maintains a table like structure where it keeps track of time range and Tablet which is serving the data within the time range. Tablet Server recovery, assignment and unassignment of new tablets to the tablet server is done in the same way as it is done in the Big Table.

We introduced a new component as "Query Processor", which is also part of the Tablet Server. Responsibility of the Query Processor is to do query processing for retrieving data from the tablets. As all the tablets managed by the tablet server have disjoint ranges, Query Processor takes advantage of this fact and does the fetching of the data in parallel from all the tablets which has the requested data. In case of pattern matching and pattern finding, Query Processor iterates over the data stored in tablet and identifies/finds the pattern of interest. Any required state for the pattern matching (ex: usage of finite state automata in KMP algorithm for string matching) can be maintained by the Query Processor.

4.2.4 Chubby

Role of Chubby in our design is similar to that of Chubby in the Big Table design. It is a central place for storing all the configurations, information about the Tablet Server which are hosting tablets, load balancing between the tablet servers, recovering from the Tablet Server failure, Master node election, helping client to find the tablet servers that

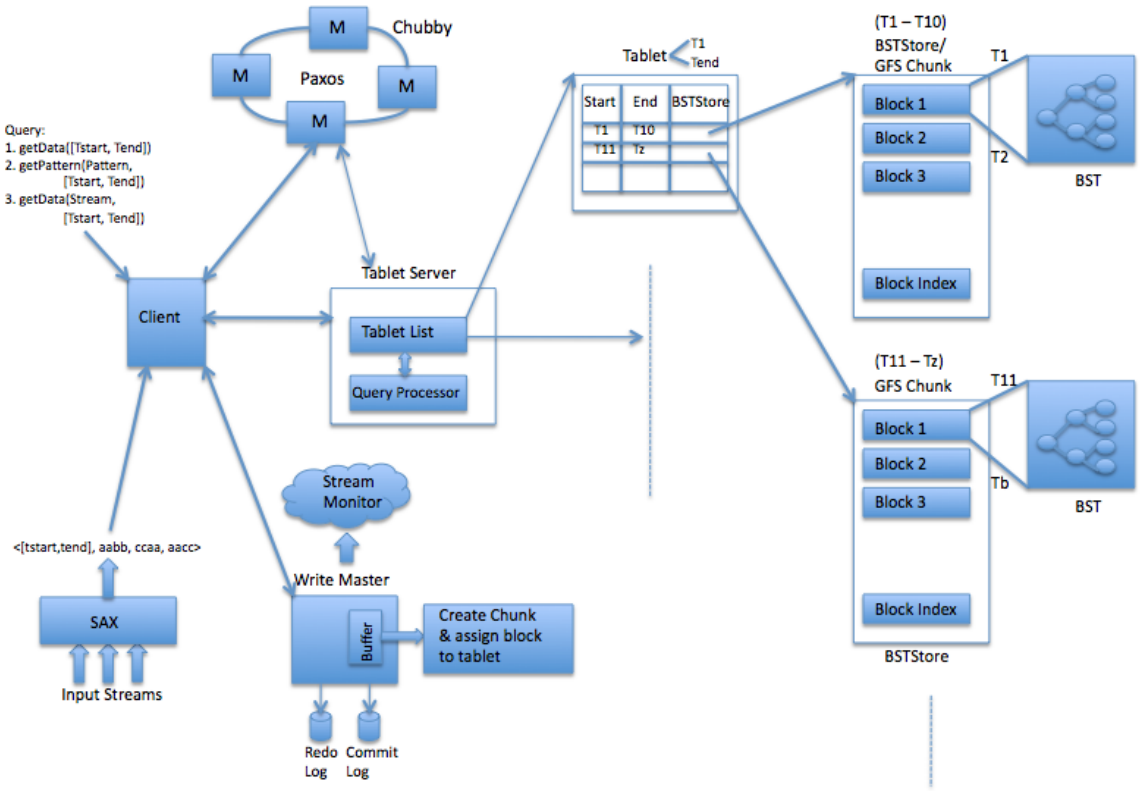

Figure 10: Design Overview.

are hosting the data for specified time range, client cache invalidation, etc. Only difference with the Google Big Table is that in our design Chubby stores the configurations for tablets in form of time range instead of key range.

4.2.5 *Write Master*

In our design we are introducing a new component called as Write Master. As mentioned in the Tablet Server section, in our design, Tablet Server is not responsible for any kind of write operation. We have simplified our design by separating out components that are responsible for read and write operations. Write Master performs all the writes to the system.

The Client sends all the data that is to be written to the system to the Write Master. Write Master maintains in memory buffer and records its all operations in redo and commit logs (which are persisted in GFS). Buffer size equals to the BSTStore (GFS Chunk size). Whenever the buffer of the Write Master gets full, a new BSTStore of size GFS Chunk is created and the system components are updated with this information. More information about this process is mentioned in subsequent section. In case of Write Master Failure, a new Write Master is appointed and in-memory buffer is created by reading redo/commit logs from GFS.

This approach has several advantages

A. Tablet Server is no more responsible for the write operations and all the tablets are read-only, hence this significantly improves systems read performance.

B. As the incoming data is buffered and asynchronously written to the persistent storage, it increases write throughput

C. For the Data Stream kind of applications, it is highly desirable to have online monitoring system and process the stream on the fly. For such cases, Write Master act as a single point of contact and all the dependent applications which need the information about the incoming data as and when it arrives, such applications can read commit/redo logs of the Write Master and provide online monitoring. Applications like d-stream [11], Time Stream [12] can very easily be integrated with our current system design.

D. As the input data is always in chronological order, creating a BSTStore from the buffer becomes a trivial operation.

4.2.6 *Client*

Client is component of our design which does all the interactions with the Chubby, Master, Tablet Server, Write Master and makes our system transparent to the external clients which are interested to persist the data in the system. Client details are mentioned in the subsequent sections while describing read/write operations.

4.3 Query (Read) and Write Operation

4.3.1 *Read Operation*

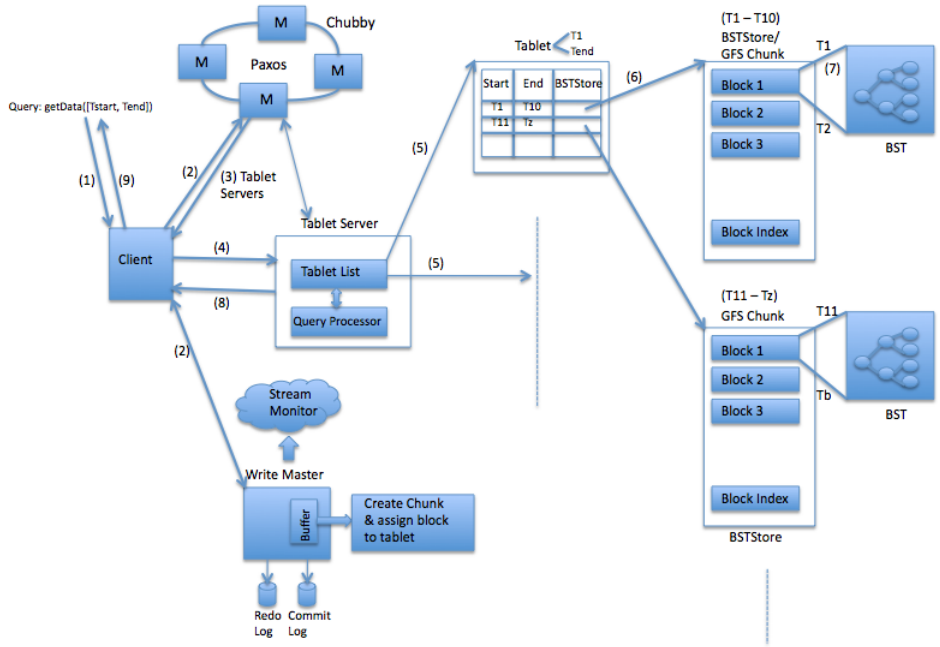

Figure 11: Read Operation Overview.

Read operation overview is provided in the figure 11. Below are the steps for basic read operation for retrieving the data from the system for a provided time range:

1. External Client calls the system API, getData([Tstart, Tend]), for retrieving all the data present for all the streams during the given time range.

2. Client contacts Master and enquires about the tablet servers that would have data in the mentioned time range. Apart from contacting Master, client would also contact Write Master in parallel, to check if the Write Master has the content in its buffer for the given time range. This is important because the external client might query for the latest data, and the data would be still present in the Write Master's Buffer and not in GFS or part of any tablet. If the write master has any data in the provided time range, it can straightway return the data to the client.

3. Master would return the list of the Tablet Server that contains the data. Client may cache this information for future queries.

4. Once the information about all the tablet servers with the time range is received by the client, the client would contact in parallel to all the tablet server and request for the data. If the number of tablet servers are way too much (for example when the client is requesting data for last one year), the client can have upper limit on the maximum connections to make to the Tablet Servers and thus avoid maintaining too many connections.

5. Incoming data query request is handled by the Query Processor component of the Tablet Server. Query Processor would do a lookup using the time range to find all the Tablets that may have data in the given time range. Once the list of Tablets having data is identified, Query Processor can fetch the data from all the tablets in parallel. Parallelism would depend upon the current load of the Tablet Server.

6. From each tablet, BSTStores which would contain the data in the given time range are identified, and corresponding GFS Chunk are fetched.

7. For each BSTStore, Block Index is loaded into memory to identify the blocks that would contain the data.

8. Data is fetched from appropriate blocks, and data is sent back to the Query Processor. Query Processor may wait for getting all the data from all the tablets or it can start replying back to the client query as and when data is fetched.

9. Client would receive the data from all the Tablet Server, and it would send back the data to the external client.

Pattern finding operation can be done in similar fashion, where Query Processor instead of sending complete data to the Client, it would implement KMP algorithm and create finite state automata for the pattern and run over the automata with the fetched data from tablets. If pattern is identified, Query Processor would return the corresponding time range.

4.3.2 Write Operation

Write operation, where the incoming data is to be persisted into our system, has two parts: Synchronous and Asynchronous Data Processing. Figure 12 provides overview of the synchronous data processing. Below are the steps involved in synchronous data processing:

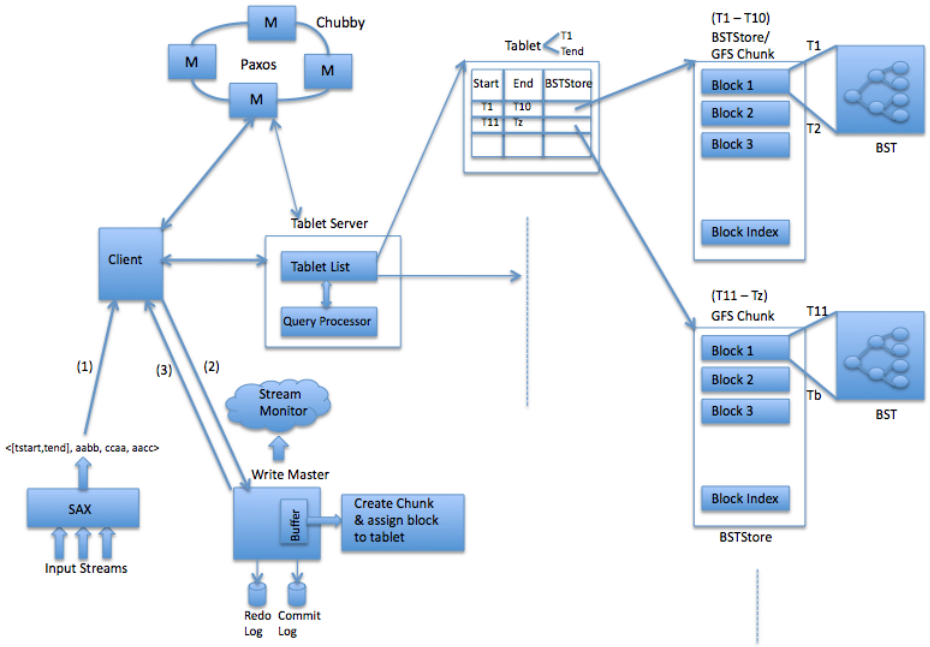

Figure 12: Write Operation Overview.

1. The module that implements SAX, generates a tuple for a window time frame combining all the stream data. Module calls the persistTuple API of the client.

2. Client sends the incoming data to the Write Master.

3. Write Master, stores the data in its in memory buffer, updates its logs for the new data and acknowledge the data receipt to the client.

As mentioned earlier, buffering of the incoming data at the Write Master improves the throughput of the system. As and when the buffer of the Write Master gets full, a background asynchronous process is initiated which does following things:

1. Creates the BSTStore from the buffered data via pushing the buffer to GFS.

2. Contacts Master and fetches the Tablet Server with the latest (unsealed) tablet. Write Master may cache this information.

3. Update the corresponding Tablet on the Tablet Server by adding a new entry with the time range and BSTStore (as created in step1) information.

4. Once update succeeds, Write Master notifies the Master about update of the time range for the tablet and tablet server, so that master in future can redirect any client request to the appropriate Tablet Server that holds the tablet information.

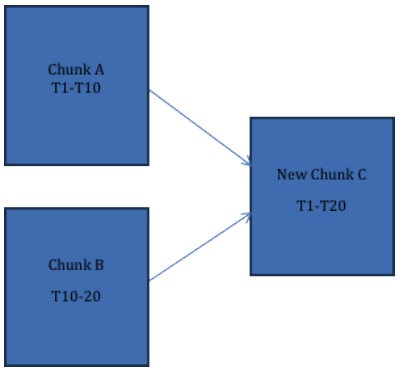

Figure 13: Compaction example where we have two old chunks Chunk A and Chunk B from which new compact Chunk C is created and corresponding entries are updated for the chunks.

4.4 Compaction

As data streams are equivalent to infinite data, the system responsible for storing data streams must have a compaction mechanism. For compaction, we can take advantage of the fact that data is stored in chronological order. The key question for compaction will be that which chunks must be selected to perform compaction. The answer to this question will be the oldest chunks because during compaction we will lose data information further as more approximation will be performed. Hence, following steps must be performed for compaction:

1. Create new chunk from two oldest consecutive chunks (The new chunk created will be equivalent to a chunk which is formed by doubling the actual pan size in window and thus, more approximation)

2. Update the corresponding tablet, tablet server and chubby master with the new time range

3. Delete old chunks

As a result we see that compaction is a necessary functionality and it shows a trade-off between the accuracy of data and storage capacity. Figure 13 summarizes the compaction process.

4.5 Handling Missing/Old Data

It is important to capture the information irrespective of the time at which it comes. We expect the arrival of input tuple to be always in chronological order. However, due to some failure or problem in the input data stream, we might miss data for a particular time period. For instance, we receive data for 6 days and don't receive any data the 3rd day. Later on, the system provides the 3rd day data and we need incorporate this data in our storage system.

The solution for the above described scenario will be to find the exact chunk location where the old data needs to be inserted. Once we know the insertion point, we break that chunk and insert a new chunk with the old data and there make necessary updates in the table, tablet server and chubby master. Another important consideration we need to take care is that as the data is immutable, and we get the same old data that is already stored then we ignore the sent old data.

4.6 Failure Handling

Our storage is a persistent store and thus, we need mechanisms for failure handling. Components which can fail include write master, tablet servers, tablets, chubby etc. As our design is highly influenced from BigTable, we use the similar failure handling mechanisms as in BigTable for tablet, tablet servers, chubby etc. In case of Write master failure, new write master is elected and it reads redo and commit logs to recreate in memory buffer and take over failed Write Master's position.

5. IMPLEMENTATION

Aim of the project was to design a system for the heterogeneous data stream processing and implementing the designed system was out of scope. Apart from the system design, as part of the project we also proposed SAX based

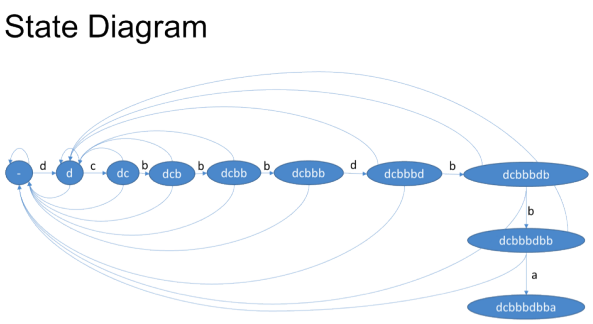

Figure 14: State diagram for pattern Query:dcbbbdbba

data model for summarizing the input data stream in the form of symbols. For doing the proof of concept of proposed data model, we implemented SAX and open sourced our SAX implementation. Implementation is available on bitbucket site on [6] url.

In traditional SAX, for converting the input stream to the symbol form, Z-score is calculated for each pane, and a symbol is assigned. In order to calculate the Z-score we need to know the mean and the standard deviation of the input stream. In traditional SAX, mean and standard deviation is calculated via considering all the points in the window. In our implementation, we did a bit divergence from the traditional approach, and maintained running mean and standard deviation for the input stream in order to capture the global trend. In order to calculate running/online mean and standard deviation, we need to maintain couple of invariant as mentioned in [7].

The results obtained from feeding the input stream to our SAX implementation to generate symbols is described in subsequent section in detail.

6. RESULT AND EVALUATION

6.1 Dataset

We generated the result by running our SAX model implementation over the MIT-BIH Polysomnographic Database [13]. The dataset contains 4 data streams EEG, BP, ECG and Respiration. Each streams generates 250 data points per sec. Window has a fixed size of 1 second and we have 5 panes per window (each pane of 0.2 sec). The symbols used for representation are *a, b, c, d*.

6.2 Pattern Query Processing

The query processing module in our system supports pattern query i.e. given a time range, finds whether a particular pattern exists in a data stream or not.

1. Query Representation: An important question regarding pattern is how will the user of system represents a query. We have provided two options for representing a query. First one is the one in which user provides query as data points. In this case, we run our SAX model over the data points and generate a string for the data points. However, it is not always possible to provide the exact query pattern data

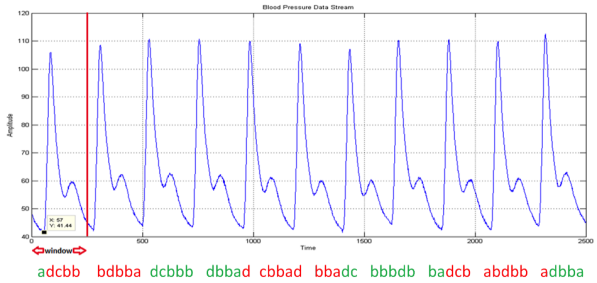

Figure 15: Output response of BP data stream from our SAX implementation

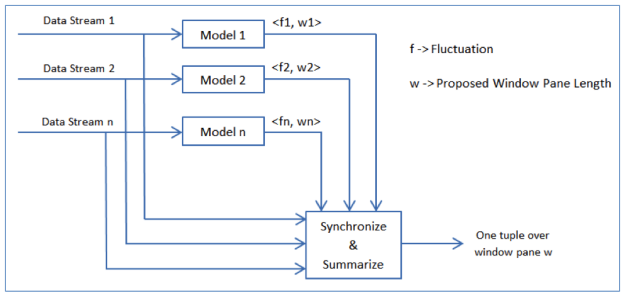

Figure 16: Adaptive Synchronization Operation Overview.

point, in that case the user can directly provide a string representation of the query.

2. Query Processing: As we represent the pattern query in form of a string, the problem of finding the pattern in the storage system boils down to string pattern matching problem. In our system, we are inclined towards using KMP [14] string matching algorithm. The motivation behind using KMP algorithm is that it creates a finite state automata (as show in Figure 14) for the pattern query. The primary advantage of building the automata is that all the data that needs to be searched on do not need be to in the main memory. Hence, we get the chunk data one at time and roll the data over the automata and if, it reaches the final state then a pattern exists.

6.3 Result

We have successfully implemented the SAX model and passed on the dataset in order to receive the string representation for the data streams (Figure 15). It has been verified that our system supports in finding pattern queries that are over multiple windows. The system is capable of running multiple pattern queries simultaneously as each pattern requires low memory requirements.

7. ADVANCED DATA MODEL: HETEROGENEOUS STREAM SYNCHRONIZATION VIA ADAPTIVE PANE LENGTH

Naive approach for handling heterogeneous data streams treats all the component data streams to be independent and assigns a constant window size for all the data streams. This assumption need not always be true. There can be important dependency relationships among the data streams. Naive approach can miss out on these dependencies. Another drawback induced by the assumption of constant pane size is that all the values inside the pane will be averaged out. Hence we might lose out on some important pattern. So it is important to address the above two drawbacks. We aim to handle these problems in our system.

7.1 How Synchronization Module Work?

Synchronization module will work closely with the summarization (SAX) module to pass data to the persistence layer. Figure 16 provides the overview of the module.

7.2 Design consideration during synchronization

Requirement for the model is that it should quantify the fluctuation for the input data stream and propose a pane length for the data stream. The proposed pane length should be such that when the input data stream is averaged over the length of window, then loss of information should be minimal. In order to achieve this goal, model should maintain history of user defined time window 'T' for the data stream, and use this history for making the proposals. [Different models can have different values of window time 'T'].

Synchronization approach can be described as follows:

7.2.1 Main Idea

Maintain 'n' models, one for each data stream. For each data stream, the model will quantify the fluctuations based on mean and standard deviation. Based on these fluctuations, the model will make a decision about the window pane length which will best capture the patterns in data with minimum loss of information.

7.2.2 Window Pane Length Computation

Each model performs the following steps for each time window to compute the ideal pane length:

A. In model, say over window length 'T', we have 'N' points.

B. On N data points, try to fit window panes of different length starting from 1 up to length N (equal to window length).

C. For each pane length, calculate the error introduced due to window approximation. [see Note for error calculation method]

D. Finalize 'W' as the maximum pane length for which error introduced over the entire window is below the permissible error rate.

E. Calculate 'Wfinal' by scaling W by window length T as: (W/N) * T

F. Approximate 'Wfinal' to the nearest integer.

G. Maintain 'Wfinal' as the proposed value for pane length for the given window.

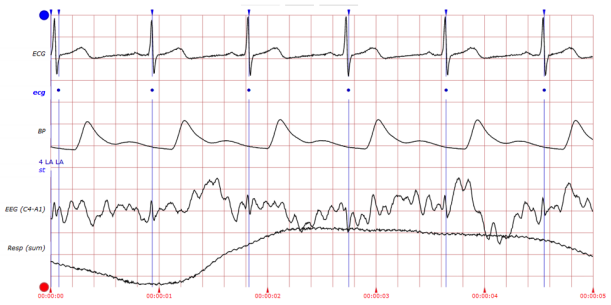

Figure 17: Heterogeneous Data Stream.

Note: For each choice of window pane, we compute a Z-score for each pane in the window. After computing the Z-score, we compute the overall error for the window by adding the 'Least Square' errors of individual panes in the window. The pane length for which the overall error is less than permissible error is chosen as the pane length for the data stream during the window in consideration.

7.2.3 Summarization with Synchronization

For each window, the summarization module takes feedback from each model about the proposed pane length for each of the data stream. From the feedback received, summarization module chooses the pane with minimum length (implies pane length of the stream with maximum fluctuations) as the pane length to be used for all the data streams. Content from all the data streams is individually averaged as per the final pane length. A single tuple is created with averaged values of all the streams to achieve synchronization.

7.3 Why does it address the identified challenges?

Consider the above heterogeneous data stream (Figure 17). From its component data streams, let us consider data streams of blood pressure (BP) and Electro-Encephalogram (EEG).

It can be observed that waveform of EEG shows greater amount of fluctuations as compared to the waveform for BP in the given time window. If Naive approach was used to process the heterogeneous data stream, then a constant pane size would be assigned for both the data streams under consideration. As a result, there is a possibility of losing out some vital information about the sharp fluctuations in EEG data stream as the data present in the waveform during the constant pane size (In the above figure, depicted by blue partition)would be averaged. The same would be case for BP data stream. In addition, the influence of sharp fluctuations in EEG on the steady rise in BP data stream will not be captured by Naive approach. This condition is also a result of the constant pane size.

If the sophisticated synchronization approach was used, then EEG would play a governing role in regards to the pane size to be used for all the data streams. As EEG has a large number of sharp fluctuations, the proposed pane size by EEG model would be quite small to capture the data as accurately as possible with minimum loss of information. Summarization module would choose the pane length proposed by EEG to be used for all the data streams. As a result, the system would not only be able to accurately capture the sharp rising and falling patterns of EEG data but also be able to able to answer queries about the pattern dependency queries about EEG and BP data stream.

8. DISCUSSION

As part of the project, we designed the system and proposed usage of SAX data model for approximation and storage of the data in order to answer large set of queries on the persisted data. Though, the implementation of the design was out of scope for the project, we did some preliminary analysis on the data model using MIT-BIH dataset [13]. With our basic experiments on the data model, we believe that SAX is a good fit .

One of the limitation or rather difficulty with the SAX data model is deciding upon the window and pane length. One of the possible solution for this could be to take values of window length and pane length as user input. In order to circumvent requirement of taking any input from user and making the system robust, in section 7, we proposed a model based approach for learning the pane length as per the fluctuations in the data stream. Though this looks like a promising approach, it is difficult to be confident about it unless we implement and do some experimental analysis.

On the system design, we did not considered handling of use case where multiple clients wants to use the same system for storing different set of heterogeneous stream data. Incorporating this requirement is not tough, and we need to add *accountname* column to all the information that we are currently storing the Chubby, Tablet Server, and Write Master. This is on the similar lines of Azure Storage System [5] where *accountname* is used for identifying data of different clients.

Though aim of our project was designing a system for the storage of heterogeneous data stream, we still believe that the job is just half way done until we implement the design and see how the system works in production environment. Most of the issues and optimization decisions are made and incorporated in design once we get basic system performance statistics and identify bottlenecks. That said, we still have lot of confidence on our design because we have built our design using Google Big Table's and Azure's design principles. As Google Big Table and Azure are used in production and are considered as pioneering storage systems, we believe that our proposed design would not have any major implementation issues.

9. CONCLUSION

As part of the project, we were able to successfully design system for storage of heterogeneous data stream, and support range queries on the persisted data as a first class read operation. Data model proposed for the storage of the data enables system to support pattern finding operation efficiently on the persisted data. Also, by adding synchronization to the heterogeneous data streams, we are able to identify patterns in each data stream more accurately. In addition, it enables the system to successfully capture any dependencies that may exist among the heterogeneous data streams. Designing the system is first step towards building

the system, and our work open up large number of avenues for future work.

10. ACKNOWLEDGMENT

We would like to thank Professor Songwu Lu for guiding us throughout the span of our project.

11. REFERENCES

[1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, Jennifer Widom *Models and Issues in Data Stream Systems* PODS 2002: 1-16

[2] Jessica Lin, Eamonn Keogh, Stefano Lonardi, Bill Chiu, *A Symbolic Representation of Time Series, with Implications for Streaming Algorithms* DMKD' 03, June 13, 2003

[3] Fay Chang, Jerey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, *Bigtable: A Distributed Storage System for Structured Data* OSDI 06

[4] Mike Burrows *The Chubby lock service for loosely-coupled distributed systems* OSDI 06

[5] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, Leonidas Rigas *Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency* SOSP '11, October 23-26, 2011

[6] *https://bitbucket.org/rohankarwa/sax*

[7] *http://en.wikipedia.org/wiki/Standarddevation*

[8] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels *Dynamo: Amazon's Highly Available Key-value Store* SOSP'07, October 14-17, 2007

[9] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel *Finding a needle in Haystack: Facebook's photo storage*

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung *The Google File System* SOSP'03, October 19-22, 2003

[11] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica *Discretized Streams: Fault-Tolerant Streaming Computation at Scale* OSP'13, Nov. 3-6, 2013

[12] Zhengping Qian, Yong He, Chunzhi Su, Zhuojie Wu, Hongyu Zhu, Taizhi Zhang, Lidong Zhou, Yuan Yu, Zheng Zhang *TimeStream: Reliable Stream Computation in the Cloud* Eurosys '13 April 15-17, 2013

[13] *http* : *//physionet.ph.biu.ac.il/physiobank/database/slpdb/*

[14] *http* : *//en.wikipedia.org/wiki/Knuth* − *Morris* − *Pratt*$_a$*lgorithm*