ANKARA YILDIRIM BEYAZIT UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND
APPLIED SCIENCES


# NOVEL MERGING BASED HEIGHT-BALANCED HISTOGRAM COMPUTATION FOR BIG DATA

M.Sc. Thesis by
TOLGA BÜYÜKTANIR

Department of Electrical and Computer Engineering

January, 2017
ANKARA

# NOVEL MERGING BASED HEIGHT-BALANCED HISTOGRAM COMPUTATION FOR BIG DATA

**A Thesis Submitted to**
**The Graduate School of Natural and Applied Sciences of**
**Ankara Yıldırım Beyazıt University**

**In Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical and Computer Engineering, Electrical and Computer Engineering**

**by**

TOLGA BÜYÜKTANIR

**January, 2017**
**ANKARA**

**M.Sc. THESIS EXAMINATION RESULT FORM**

We have read the thesis entitled **"NOVEL MERGING BASED HEIGHT-BALANCED HISTOGRAM COMPUTATION FOR BIG DATA"** completed by **TOLGA BÜYÜKTANIR** under supervision of **ASSIST.PROF.AHMET ERCAN TOPCU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist.Prof.Dr. Ahmet Ercan Topcu

Supervisor

Assist.Prof.Dr. M. Abdullah Bülbül

Jury Member

Assist.Prof.Dr. A. Osman Çıbıkdiken

Jury Member

Prof.Dr. Fatih V. ÇELEBİ

Director

Graduate School of Natural and Applied Sciences

## ETHICAL DECLARATION

I hereby declare that, in this thesis which has been prepared in accordance with the Thesis Writing Manual of Graduate School of Natural and Applied Sciences,

- All data, information and documents are obtained in the framework of academic and ethical rules,
- All information, documents and assessments are presented in accordance with scientific ethics and morals,
- All the materials that have been utilized are fully cited and referenced,
- No change has been made on the utilized materials,
- All the works presented are original,

and in any contrary case of above statements I accept to renounce all my legal rights.

Tolga BÜYÜKTANIR

## ACKNOWLEDGMENTS

This thesis could not have been written without my advisor, Asst. Prof. Ahmet Ercan Topcu, who encouraged, motivated and supported me always and everywhere through my thesis term. I would like to thank him for his guidance and patience throughout the preparation and execution of this research.

I would like to thank Fatih Emekci, Burak Yıldız and Erdem Özdemir who spent their valuable time to my thesis, supported me without expectation. Without them and without their guidance and friendly attitudes, I would not complete this thesis.

I would like also thank my family, lecturers, colleagues, homemate Cumhur Kuşoğlu and friends who supported and encouraged me.

Finally, I would like to thank everyone who could not be mentioned in this section, but prayed for me.

## NOVEL MERGING BASED HEIGHT-BALANCED HISTOGRAM COMPUTATION FOR BIG DATA

## ABSTRACT

The amount of data generated and stored in cloud systems has been increasing exponentially. The examples of data include user generated data, machine generated data as well as data crawled from the Internet. There have been several frameworks with proven efficiency to store and process the petabyte scale data such as Apache Hadoop ecosystem tools, and several NoSQL frameworks. These systems have been widely used in industry and thus are subject to several research. The proposed data processing techniques should be compatible with the above frameworks in order to be practical.

One of the key data operations is deriving height-balanced histograms as they are crucial in understanding the statistical properties of the underlying data with many applications including query optimization.In this thesis, we focus on approximate height-balanced histogram construction for big data and propose a novel merge based histogram construction method with a histogram processing framework which constructs an height-balanced histogram for a given time interval. The proposed method constructs approximate height-balanced histograms by merging exact height-balanced histograms of partitioned data by guaranteeing a maximum error bound on the number of items in a bucket (bucket size) as well as any range on the histogram. We also test Apache Pig User Define Functions of this proposed method in this thesis.

**Keywords :** approximate histogram, merging histograms, big data, log files

## NOVEL MERGING BASED HEIGHT-BALANCED HISTOGRAM COMPUTATION FOR BIG DATA

## ÖZ

Üretilen ve bulut sistemlerde kaydedilen data miktarı her geçen gün katlanarak artmaktadır. Buna örnek olarak, kullanıcı tarafından üretilen veriler, makine tarafından üretilen veriler ve İnternet'ten crawl edilen veriler gösterilebilir. Petabyte boyutunda dataları depolamak ve işlemek için; Apache Hadoop ekosistem araçları ve bazı NoSQL frameworkleri gibi verimliliği kanıtlanmış frameworkler vardır. Bu araçlar endüstride geniş çaplı kullanılmaktadır ve bu sebepten çeşitli araştırmalara konu olmaktadır. Önerilen veri işleme teknikleri yukarıda saydığımız frameworklere pratik olması için uyumlu olmalıdır.

Önermli veri operasyonlarından bir tanesi de, height-balanced(yada equi-depth) histogram oluşturmaktır. Çünkü equi-depth histogramlar, sorgu optimizasyonu da gerektiren birçok uygulamada, datanın istatistiksel özelliğini anlamak için hayati öneme sahiptir. Bu tezde, büyük veriler için approximate equi-depth histogramının oluşturulması üzerine çalışılmıştır ve verilen zaman aralığının equi-depth histogramını oluşturan histogram birleştirme tabanlı yeni bir metod ve bu metodu kullanan bir framework geliştirilmiştir. Bu framework, parçalar halinde bulunan tam olarak hesaplanmış equi-depth histogramları birleştirmek kaydıyla yaklaşık bir equi-depth histogram oluşturmaktadır. Oluşturulan bu histogramın bir bucketında bulunan öğe sayısınında oluşabilecek maksimum hata sınırı garanti edilmektedir. Histogramın herhangi bir aralığında da maksimum hata sınırı garanti edilmektedir. Biz bu tezde önerdiğimiz metodun Apache Pig ve web uygulamalarını da sunmaktayız.

**Keywords :** approximate histogram, merging histograms, big data, log files

CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## NOMENCLATURE

| | |
|---|---|
| $\beta$ | Number of Buckets |
| $T$ | T-buckets |
| $H^0$ | Pre-histogram after first assembling of $k$ exact equi-depth histograms, $H_1$, $H_2$, ..., and $H_k$ |
| $H^*$ | Final approximate equi-depth histogram after bucket merging operations of $H^0$ |
| $H^e$ | Exact equi-depth histogram for the union of $k$ value sets, $P_1$, $P_2$, ..., and $P_k$ |
| $s(i, H)$ | The $i^{th}$ bucket size of the equi-depth histogram $H$ |
| $a(i, H)$ | The $i^{th}$ approximate bucket size of the approximate equi-depth histogram $H$ |
| $S(i, H)$ | The $i^{th}$ cumulative size of the equi-depth histogram $H$ |
| $A(i, H)$ | The $i^{th}$ approximate cumulative size of the equi-depth histogram $H$ |
| $R(i, j, H)$ | Sum of bucket sizes starting from the $i^{th}$ bucket up to the $j^{th}$ bucket (both inclusive) of the equi-depth histogram $H$ |

# CHAPTER 1

## INTRODUCTION

The data generated and stored by enterprises are in the orders of terabytes or even petabytes [1–3]. We can classify the source of the data in the following groups : machine generated data (a.k.a logs), social media data, transactional data and data generated by medical and wearable devices. Processing the produced data and deriving results are critical in decision making and thus the most important competitive power for the data owner. Therefore, handling such big datasets in an efficient way is a clear need for many institutions. Hadoop MapReduce [4, 5] is a big data processing framework that has rapidly become the standard method to deal with data bombarding in both industry and academia [3, 6–10]. The main reasons of such strong adoption are the ease-of-use, scalability, failover and open-source properties of Hadoop framework. After the wide distribution, many research works (from industry and academia) have focused on improving the performance of Hadoop MapReduce jobs in many aspects such as different data layouts [8, 11, 12], join algorithms [13–15], high-level query languages [3, 7, 10], failover algorithms [16], query optimization techniques [17–20], and indexing techniques [6, 21, 22].

In today's fast-paced business environment, obtaining results quickly represents a key desideratum for *Big Data Analytics* [8]. For most applications on large datasets, performing careful sampling and computing early results from such samples provide a fast and effective way to obtain approximate results within the predefined level of accuracy. The need for approximation techniques grow with the size of the data sets and most of the time they shed a light to make fast decisions for the businesses. General methods and techniques for handling complex tasks have room to improve in both MapReduce systems and parallel databases. For example, consider a web site, such as a search engine, consists of several web server hosts ; user queries (requests) are collectively handled by these servers (using some scheduling protocol) ; and the overall performance of the web site is characterized by the latency (delay) encountered by the users. The distribution of

the latency values is typically very skewed, and a common practice is to track some particular quantiles. The Yahoo website, for instance, handles more than 43 million hits per day [23], which translates to 40000 requests per second. The Wikipedia website handles 30000 requests per second at peak, with 350 web servers. While all three questions relate to computing of statistics over data, they have different technical nuances, and often require different algorithmic approaches as accuracy can be traded for performance.

One way to obtain statistical information from data is histogram construction. Histograms summarize the whole data and give information about distribution of the data. Moreover, the importance of histogram increases when the size of the data is huge. Since, histograms are very useful and are efficient ways to get quick information about data distribution, they are highly used in database systems for query optimization, query result estimation, approximate query answering, data mining, distribution fitting, and parallel data partitioning [24]. One of the most used histogram types in database systems is the equi-depth histogram. The equi-depth histogram is constructed by finding boundaries that split the data into a predefined number of buckets containing equal number of tuples. More formally, $\beta$-bucket equi-depth histogram construction problem can be defined as follows : given a data set with $N$ tuples, find the boundary set $B = b_1, b_2, \ldots, b_{\beta-1}$ that splits the sorted tuples into $\beta$ buckets, each of which has approximately $N/\beta$ tuples.

In this thesis, we propose a framework to compute equi-depth histograms on-demand (dynamic) from the precomputed histograms of the partitioned data. In order to do so, we propose a histogram merging algorithm giving a user specified error bound on the bucket size. In particular, we merge $T$-bucket histograms to build a $\beta$-bucket histogram for the underling data of size $N$ and give a mathematical proof showing $2\beta/T$ error rate on the bucket size and as well as any range on the histogram. In our framework, users specify $T$ and $\beta$, we compute $T$-bucket histograms for each partition, and a query asking for a histogram of any subset of the partitions. Then, the framework computes the $\beta$ histogram on-demand from the offline computed histograms of the partitions. In real life systems, the pre-computation is done incrementally (i.e., daily, hourly or monthly) such as

logs and database transactions. We began all of this work inspired by patent of Emekci et al. [25].

Our contribution can be summarized as follows :

- We proposed a novel algorithm to build an approximate equi-depth histogram for a union of partitions from the sub-histograms of the partitions.
- We theoretically and experimentally showed that the error percentage of a bucket size is bounded by a predefined error set by the user (i.e., $\varepsilon_{max}$).
- We theoretically and experimentally showed that the error percentage of a range size is bounded by a predefined error set by the user (i.e., the same above $\varepsilon_{max}$).
- We implemented our algorithm on Hadoop and demonstrated how to apply it to practical real life problems.

The rest of the thesis is organized as follows : In Section 2, related works are summarized. In Section 3, we introduce the histogram construction problem for big data. We have exemplified related histogram types, and we give some background information about cloud computing environments in Section 4. In Section 5, in-depth explanation of the proposed method takes place. The details of implementation on Hadoop MapReduce framework is given in Section 6. Evaluation methodology and experimental results are discussed in Section 7. We present implementation of our method to Apache Pig, a demo application and experimentations of demonstrations in Section 8, and finally we conclude the thesis with Section 9.

# CHAPTER 2

## RELATED WORKS

Basically, histograms are used to get quick distribution of information from the given data. This quick information is used especially in database systems in computer science e.g. selectivity estimation to optimize queries, load balancing of join queries, and much more [24]. There are different types of histograms and each type of histogram has different properties [26]. Exact histogram construction is not feasible when the data is too big or the data is frequently updated. In such cases, histograms are constructed from sampled data and/or maintained according to the updated data [27]. This type of histograms are called approximate histograms rather than exact histograms. Approximate histogram construction from sampled data can be divided into two categories by sampling method [28] which are tuple-level sampling and block-level sampling. Tuple-level sampling method uses uniform-random-sampling to sample the data at tuple level to construct an approximate histogram at the desired error bound [29, 30]. Gibbons et al. [29] proposed a sampling-based incremental maintenance approach of approximate histograms. The proposed approach, backing sample, keeps the sampled tuples up-to-date in a relation. A bound of the amount of the sampling size for a given error bound studied by Chaudhuri et al. [30] in addition to proposing an adaptive page sampling algorithm. The second method, block-level sampling, exemplifies the data according to an iterative cross-validation based approach [31, 32]. Chaudhuri et al. [32] proposed a method for approximate histogram construction using an initial set of data and iteratively updated the constructed histogram until the histogram error is under the predetermined level. All of the proposed approaches above, however, are for single-node databases.

When the data is too big to handle in a single-node database, the data is distributed to multi-nodes. One of the well-known distributed data storage frameworks is Hadoop Distributed File System (HDFS) [33] and the data processing framework of the stored data in the HDFS is Hadoop MapReduce [5]. The histogram construction of such distributed data is not well-studied and there is less work

on histogram creation of distributed data than the ones on undistributed data. One of the adapted methods for contructing approximate histogram is tuple-level sampling. Okcan et al. [15] proposed a tuple-level sampling based algorithm to construct approximate equi-depth histograms for distributed data to improve processing theta-joins using MapReduce. The algorithm works as follows. In the map section of a MapReduce Job, a predefined number of tuples are selected randomly by scanning the whole data and outputted. The tuples are sorted and sent to the reducer. The reducer of the job determines and outputs the boundaries of equi-depth histograms. In [34], a method for approximate wavelet histogram construction for big data using MapReduce is proposed and an improved sampling method -ignoring low frequent sampled keys in splits- is given. The drawback of such histogram construction algorithms of distributed data using tuple-level sampling is that scanning the whole data is a time consuming process. Another approximate histogram construction method is proposed in [28]. This method also uses a sampling method named two-phase sampling which samples the whole data at block-level and constructs the approximate histogram and calculates the error. If the error is not in the desired error boundary, the additional sampling size needed is calculated and histogram construction process is repeated. The insufficiency of this method is that histogram is rebuilt for every new data and it requires a customized MapReduce framework. In this paper, we propose a novel approximate equi-depth histogram construction method with a log histogram monitoring framework that users can query the daily stored log files for their equi-depth histogram. In the proposed method, a MapReduce Job is scheduled to summarize the daily stored log files which means that the exact equi-depth histogram of each log file is constructed and stored in corresponding summary files and another MapReduce Job merges the summaries of intended log files for approximate equi-depth histogram construction.

# CHAPTER 3

## PROBLEM DEFINITION

In this section, we motivate the problem with a practical example and then formally define it. Machine generated data, also known as logs, is automatically created by machines. Logs contain list of activities of machines. In general, logs are stored daily in W3C format [35]. When we consider a web server, requests from web applications and responses from the server are written to log files. There are several actors deriving intelligence from these logs. For example ; operations engineers derive operational intelligence (response times, errors, exceptions etc.) and business analyst derives business intelligence (top requested pages, top users, click through rates etc.). In the context of web applications, the need to analyze clickstreams has increased rapidly, and in order to answer the demand, businesses build log management and analytical tools. A typical internet business may have thousands of web servers logging every activity on the site. In addition, they have ETL processes incrementally collecting, cleaning and storing the logs in a big data storage (i.e., This is usually Hadoop and its storage HDFS). This work-flow is demonstrated in Figure 3.1. The amount of data to ETL and to run analytics on is huge and has been increasing rapidly. Most of the time, customers would be happy to trade accuracy for performance as they need a quick intelligence to make fast decisions. One quick and reliable way to understand the statistics about the underlying data is using equi-depth histograms. As the paper [36] we have written on this subject shows, we outline a framework computing on-demand histograms of the data for any time interval for the above scenario. In web servers, daily logs are kept instantly. At the end of the day, all the log files belonging to that day are concatenated in a single log file and it is pushed to HDFS. As soon as the new log file is available in the HDFS, an exact equi-depth histogram is built and stored in the HDFS in a new summary file by the Summarizer Job. This means that the equi-depth histogram of each daily log is stored. Then, if a histogram for any time interval is requested (for example histogram for the last month), the Merger Job fetches an equi-depth histogram of each histogram and merges them 3.2 using the proposed merging algorithm explained in the following sections. We

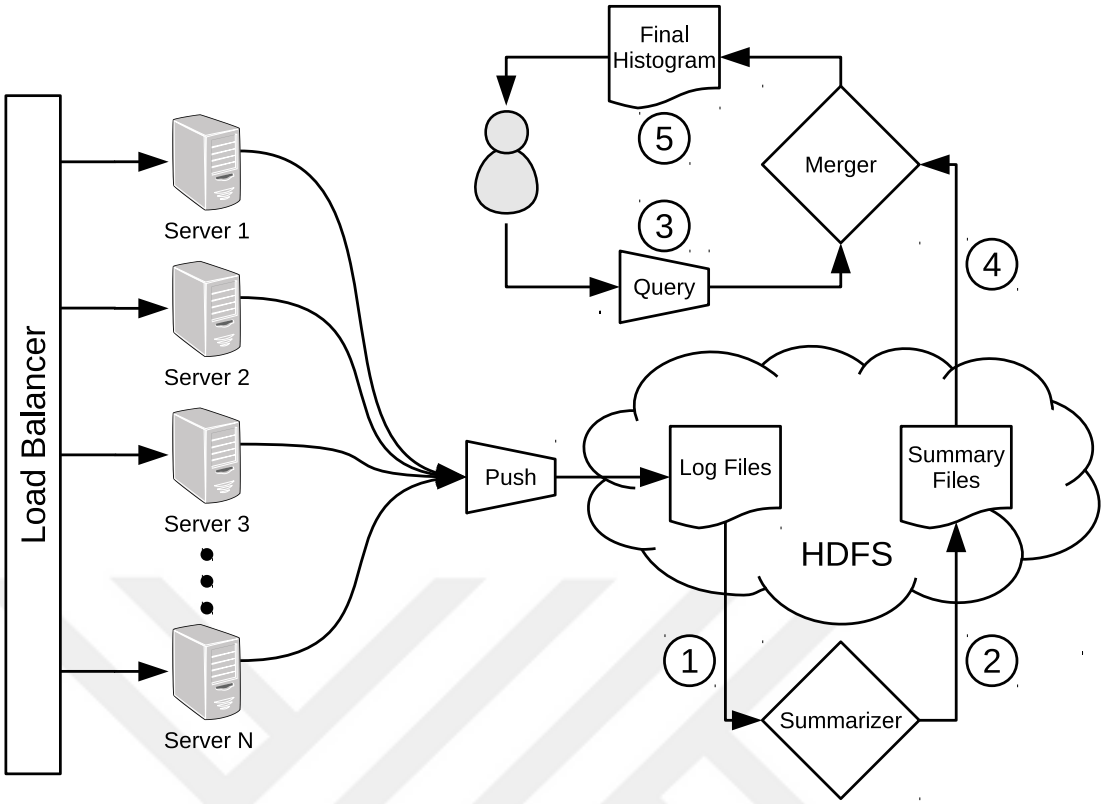



**Figure** 3.1 Flowchart of the proposed method

also provide an error rate on the histogram in order to increase the confidence. Although we motivate our framework with logs, it can be applied without loss of generality to any structured data where we need a histogram such as database transactions, etc...

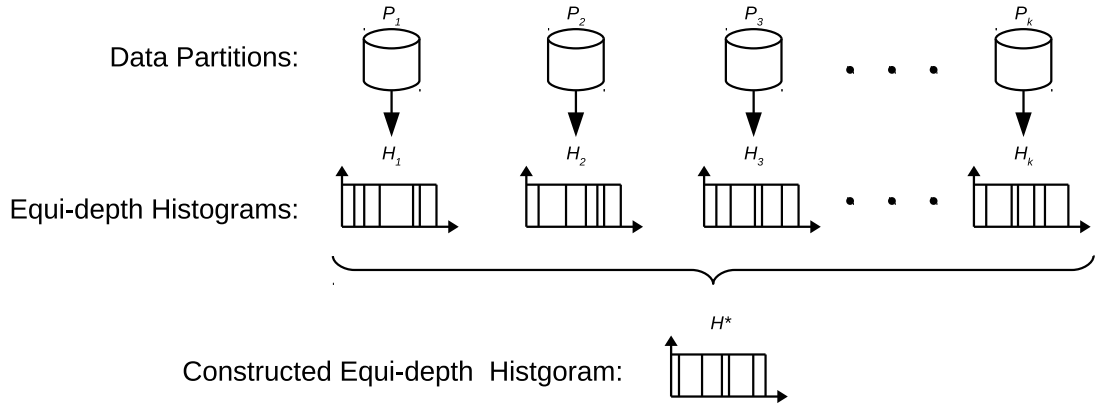



**Figure** 3.2 Histogram building by merging exact histograms of data partitions

After motivating and showing the need, we can formulate the problem we are solving as follows :

***Problem Definition :*** Given $k$ partitions, $P_1, P_2, ..., P_k$, and their respective $T$-bucket equi-depth histograms, $H_1, H_2, ..., H_k$, build a $\beta$-bucket equi-depth histogram $H^*$ where $\beta \leq T$ over $P_1, P_2, ..., P_k$ where $|P_1| + |P_2| + ... + |P_k|$ is equal to $N$ and $B_1, B_2, ..., B_\beta$ are the buckets of $H^*$ such that :

- The size of any bucket $B_i$ is $(N/\beta) \pm \varepsilon_{max}$ where $\varepsilon_{max} < 2\beta/T \times (N/\beta)$.
- The size of any range spanning $m$ buckets $B_i$ through $B_j$ is $m \times (N/\beta) \pm \varepsilon_{max}$ where $\varepsilon_{max} < 2\beta/T \times (N/\beta)$.

# CHAPTER 4

## BACKGROUND

In this section, some background information is given about cloud computing environments such as Distributed File System (DFS) [37], MapReduce (MR) [38], Pig [39], and Hive [40].

***Distributed File System (DFS) :*** DFS is a generalized name of distributed, scalable, and fault-tolerant file systems such as Google FS [37] and Hadoop DFS [33]. In particular, we address the HDFS in this paper [36] and thesis. In HDFS, large files are divided into small chunks and these small chunks are replicated and stored in multiple machines named DataNodes. The replication process ensures that HDFS is fault-tolerant. The metadata of the stored files such as name, replication count, file chunk locations, etc. are indexed in NameNode which is another machine. Clients read and write files to HDFS by interacting with the NameNode and the DataNodes.

The overall system architecture of HDFS is seen in Figure 4.1. In the figure, the NameNode takes place at the top and the DataNodes at the bottom. The replicas of the file chunks are labeled with the same numbers. The NameNode can interact with the DataNodes to maintain the file system by controlling the health and balancing the loads of the DataNodes. If there is a problem in a

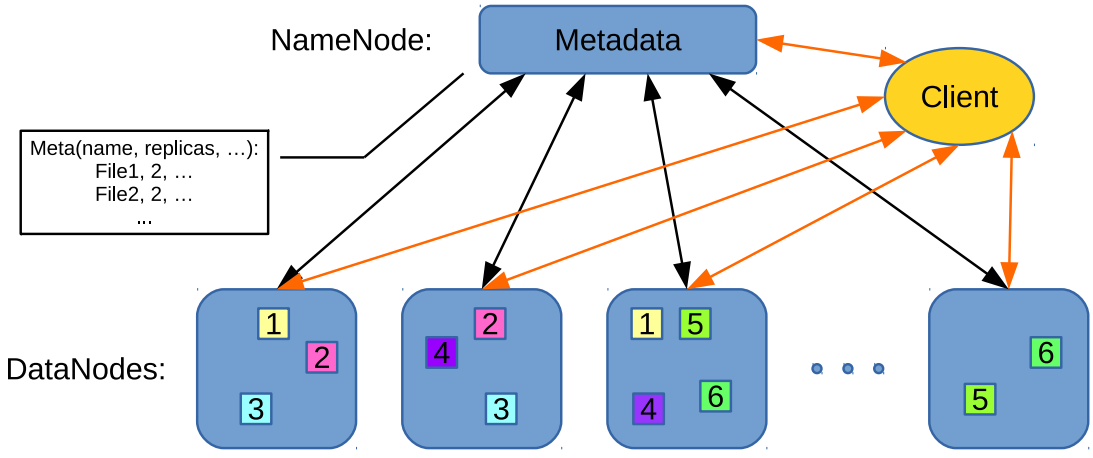



**Figure** 4.1 General architecture of Hadoop Distributed File System

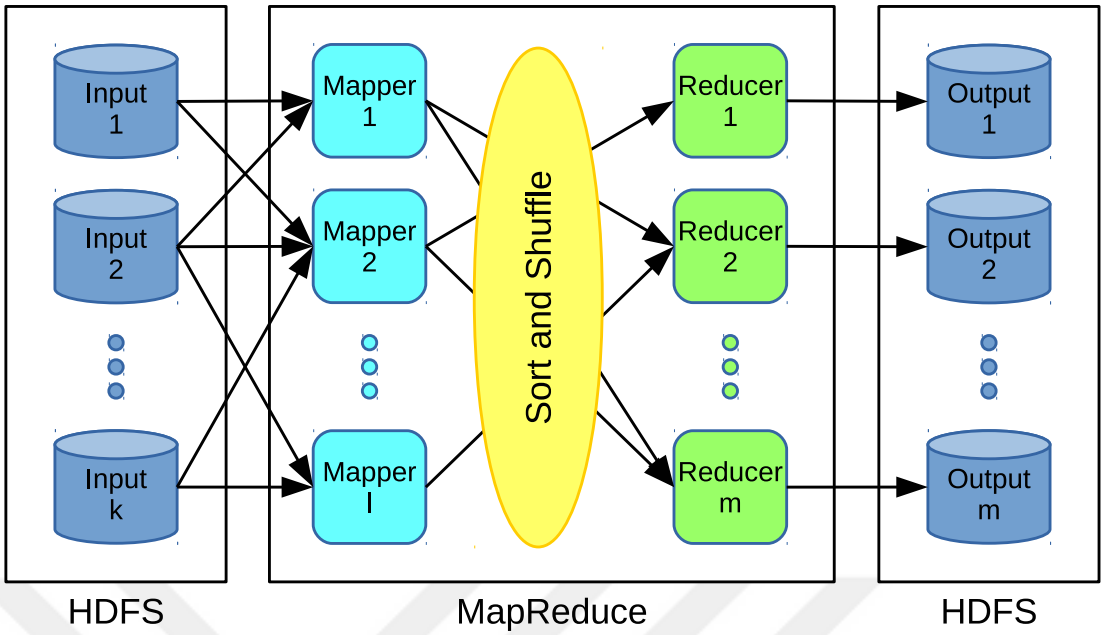



**Figure** 4.2 Overview of Hadoop MapReduce Framework

DataNode, the NameNode detects the problematic DataNode and replicates the file chunks in that DataNode to other DataNodes. Clients can also interact with the whole HDFS to read existing files and write new files to HDFS.

***MapReduce (MR) :*** MapReduce is a programming model for processing huge datasets which is especially resided in distributed file systems. Besides, MapReduce framework is the combination of the components which executes submitted MapReduce tasks by managing all resources and communications among the cluster while providing for fault tolerance and redundancy. In this paper [36] and thesis, we specifically handle the Hadoop MapReduce framework.

A MapReduce task consist of Mappers and Reducers. The Mapper has a method called Map which gets $\langle key, value \rangle$ pairs as input and emits $\langle key, value \rangle$ pairs as intermediate output. The intermediate output is shuffled and sorted by a component of the MapReduce framework at the Sort and Shuffle phase and all $\langle key, value \rangle$ pairs are grouped and sorted by keys at this phase. The output of the Sort and Shuffle phase is $\langle key, [value1, value2, ...] \rangle$ pairs and this is the input of the Reduce method which is in the Reducer. After the Reduce method finishes it's job, it also emits $\langle key, value \rangle$ pairs as final output of the MapReduce task. In some cases, a Combiner is also included in MapReduce tasks which is often the

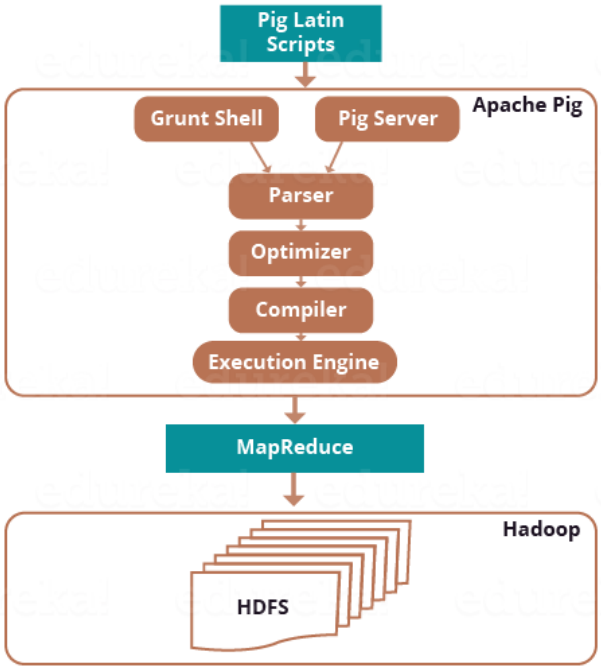



**Figure** 4.3 Architecture of Apache Pig

same with the Reducer. The Combiner has a Combine method which combines the output of the Map method to decrease network traffic.

The summary picture of the MapRecude framework is given in Figure 4.2. In the figure, input to a Mapper is read from HDFS. The output of the Mappers goes through the Sort and Shuffle phase and Reducers get the sorted and shuffled data and process it and write the output to HDFS, again.

***Apache Pig :*** Pig is a platform for processing big data with query programs written in a procedural language called Pig Latin. Query programs are translated into MapRecude tasks and the tasks are run over MapReduce framework. The queries can be written by using both existing and user defined functions. Thus, Pig is an extensible platform and users can create their own functions. The Apache Pig architecture which is taken from *edureka* is demonstrated in Figure 4.3

***Apache Hive :*** Hive is another platform for storing and processing large data-sets like Pig. Hive has its own SQL-like declarative querying language named as HiveQL. HiveQL also supports custom user defined Map/Reduce tasks in queries. The Apache Hive architecture which is taken from *edureka* is demonstrated

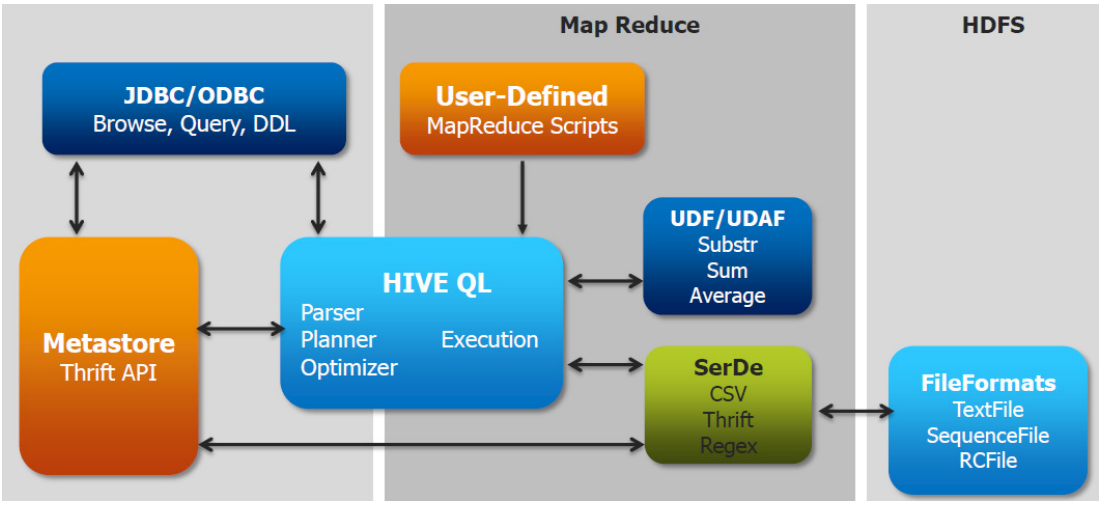



**Figure** 4.4 Architecture of Apache Hive

**Table** 4.1 An example data distribution

| Number of Employees | Salary(thousand Turkish Lira) |
|---|---|
| 5 | 1.0 |
| 3 | 2.0 |
| 7 | 2.5 |
| 10 | 5.0 |
| 2 | 6.5 |
| 1 | 9.0 |

in Figure 4.4.

## 4.1 Histogram Types

Histogram construction is one way to obtain statistical information from data. Because histogram give information about data distribution and it is a summary all of the data. Moreover, histogram provide quick query optimization, approximate result, distribution fitting and parallel data partitioning [24]. Therefore, error guarantees, time and accuracy performance are expected. In this section, we explain histogram types with a peace of examples with table 4.1 which contains salary informations and its frequency.

***Trivial Histogram :*** The simple histogram type is based on uniform distribution. It consists of a buckets. In Figure 4.5, we collect all values displayed in


**Figure** 4.5 Trivial Histogram


**Figure** 4.6 Equi-Width Histogram

Table 4.1 in single bucket. This histogram approach has maximum error rate. Because, according to Figure 4.5, when we runs SQL command showed in Listing 4.1, we gets 28 values at the worst case. But we know we have 3 value greater than 6000.

Listing 4.1: Query 1

```
select count() from Employee where salary 6000
```

***Equi-Width Histogram :*** This histogram type divide value axis to equal ranges. The example histogram is available in Figure 4.6. Each bucket shows frequency of its range. In our example query which is in Listing 4.1, we have 3 values. But, according to Figure 4.6, there are 11 values. Disadvantages of this histogram, there are high variance and error rate estimation is difficult.


**Figure** 4.7 Equi-Depth Histogram

***Equi-Depth Histogram :*** Equi-depth histogram is called as equi-heigh and heigh-balanced histograms. In Figure 4.7, Equi-depth histogram of given data distribution is constructed. According to this histogram, we can predict 7 values is greater then 6000 at the worst case. Equi-depth histogram preserves order of values, but it has a disadvantage which is sometimes variance in a bucket may be high.

***V-optimal histograms :*** Another histogram type is V-optimal histograms. V-optimal histograms optimize variance. It is also called as variance-optimal histogram. The aim of this histogram is minimize the variance of each bucket. In this section, we didn't give an example for this histogram type.

# CHAPTER 5

## EQUI-DEPTH HISTOGRAM BUILDING

In this section, we explain our approximate equi-depth histogram construction method in detail. In the first part of the method, exact-equi depth histograms of data partitions are constructed. This part is done offline with a well-know straight-forward histgoram construction algorithm. In the second and the important part of the method, equi-depth histograms are merged to construct an approximate equi-depth histogram over the partitions. One important feature is that the constructed histogram comes with maximum error bound on both size each bucket and size of any bucket range.

In the following part of the section, merging part of the method is explained with an example and then the algorithm of the merging is given and the section is concluded with maximum error bound theorems and their proofs.

A $T$-bucket equi-depth histogram $H$ for a set of values $P$ (may be called a partition) can be described as an increasing sequence of numbers, which represents the boundaries. Each pair of consecutive boundaries defines a bucket, and the size of this bucket is the number of values between its boundaries, where inclusive at the front and exclusive at the end (except the last bucket). Last bucket size also includes the last boundary. For an exact equi-depth histogram, size of each bucket is the same and equals and exactly total number of values divided by total number of buckets. On the other hand, bucket sizes of an approximate equi-depth histograms may not be equal.

We express a $T$-bucket equi-depth histogram as$H = \{(b_1, s_1), (b_2, s_2), \ldots, (b_i, s_i), \ldots, (b_{T-1}, s_{T-1}), (b_T, 0)\}$, where $b_i$ indicates the $i_{th}$ boundary and the $s_i$ indicates the $i^{th}$ bucket size for exact histograms (the approximate size of the $i^{th}$ bucket for approximate histograms), for the rest of the paper. Let us have two example value sets, $P_1$ and $P_2$, which are $\{2, 4, 5, 6, 7, 10, 13, 16, 18, 20, 21, 25\}$ and $\{3, 9, 11, 12, 14, 15, 17, 19, 22, 23, 24, 26, 27, 29, 30\}$. According to the value sets, $|P_1|$ and $|P_2|$ which represent number of values in each set, equal to 12 and 15, res-

pectively. 3-bucket histogram of $P_1$ is $H_1 = \{(2,4),(7,4),(18,4),(25,0)\}$ and $P_2$ is $H_2 = \{(3,5),(15,5),(24,5),(30,0)\}$ and graphical representation of them are given in Figures 5.1 and 5.2. First bucket of $H_1$ contains the first four values, $\{2,4,5,6\}$, the second bucket contains four values, $\{7,10,13,16\}$, and the third (also the last) bucket contains the last four values $\{18,20,21,25\}$. For $H_2$, first bucket has five values, $\{3,9,11,12,14\}$, the second bucket contains five values, $\{15,17,19,22,23\}$, and the last bucket has five values, $\{24,26,27,29,30\}$. Let us define a $s(i,H)$ function which denotes the size of $i^{th}$ bucket of the equi-depth histogram $H$, and a $S(i,H)$ function which denotes the cumulative size of all buckets from the first to $i^{th}$ bucket of $H$, that is,

$$S(i,H) = s(1,H) + s(2,H) + \cdots + s(i,H) \tag{5.1}$$

Then, the convention assures that $S(i,H) = i \times |P|/T$, for all $i \leq T$, where $|P|$ is the number of values and $T$ is the number of buckets. Considering $H_1$, $s(1,H_1) = s(2,H_1) = s(3,H_1) = 4$. For cumulative sizes, $S(1,H_1) = 4$, $S(2,H_1) = 8$, and $S(3,H_1) = 12$. Bucket sizes of $H_2$ is $s(1,H_2) = s(2,H_2) = s(3,H_2) = 5$ and cumulative sizes are $S(1,H_2) = 5$, $S(2,H_2) = 5$, and $S(3,H_2) = 5$.

Let us define two more functions, $a(i,H)$ and $A(i,H)$, which are the $i^{th}$ approximate bucket size and the $i^{th}$ cumulative bucket size for approximate equi-depth histograms, respectively. By writing an approximate version of Equation 5.1, we get the following equation :

$$A(i,H) = a(1,H) + a(2,H) + \cdots + a(i,H) \tag{5.2}$$

Lastly, let us define a range function $R(i,j,H)$ that gives the sum of sizes of buckets which starts from the $i^{th}$ bucket up to the $j^{th}$ bucket, both inclusive. The formal definitions are given in the following formulas for both exact bucket sizes and approximate bucket sizes.

$$R_s(i,j,H) = s(i,H) + s(i+1,H) + \cdots + s(j,H) \tag{5.3}$$

$$R_a(i,j,H) = a(i,H) + a(i+1,H) + \cdots + a(j,H) \tag{5.4}$$

The definitions given belove are summarized in Table 5.1.

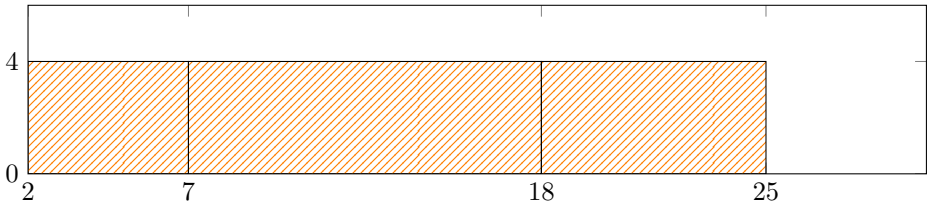



**Figure** 5.1 A sample equi-depth histogram $H_1$ with 3 buckets, based on data $\{2, 4, 5, 6, 7, 10, 13, 16, 18, 20, 21, 25\}$.

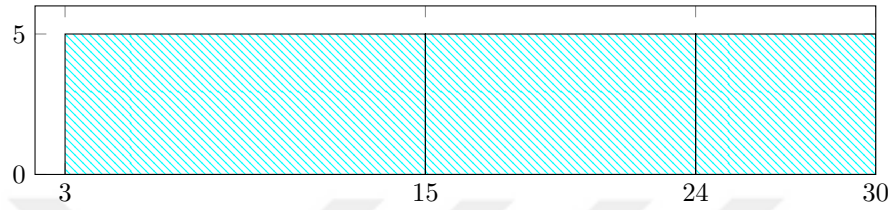



**Figure** 5.2 Another sample equi-depth histogram $H_2$ with 3 buckets, which represents data $\{3, 9, 11, 12, 14, 15, 17, 19, 22, 23, 24, 26, 27, 29, 30\}$.

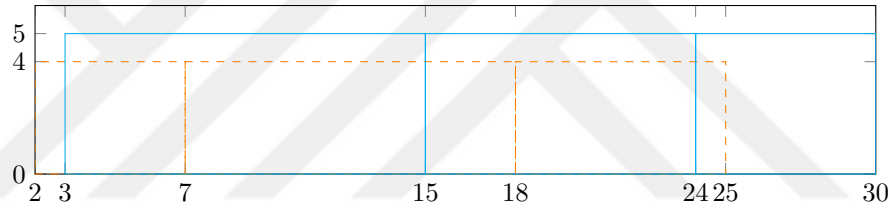



**Figure** 5.3 Example equi-depth histograms given in Figures 5.1 (orange-dashed) and 5.2 (cyan-solid) are coupled together, with boundary sequence $\{2, 3, 7, 15, 18, 24, 25, 30\}$.

Since we completed the definitions for convention, we start to explain the merging process in detail. We have exact 3-bucket equi-depth histograms $H_1$ and $H_2$ given in Figures 5.1 and 5.2 for the example value sets $P_1$ and $P_2$ where $P_1 = \{2, 4, 5, 6, 7, 10, 13, 16, 18, 20, 21, 25\}$ and $P_2 = \{3, 9, 11, 12, 14, 15, 17, 19, 22, 23, 24, 26, 27, 29, 30\}$. The total number of values $N$ is equal to $|P_1| + |P_2| = 12 + 15 = 27$ and let bucket count of final histogram, $\beta$, be 3. As seen in the histograms $H_1$ and $H_2$, $H_1$ has a boundary sequence of $2, 7, 18, 25$ and each $H_1$ bucket has $12/3 = 4$ values, $H_2$ has $3, 15, 24, 30$ and bucket size of $15/3 = 5$. In Figure 5.3, we show $H_1$ and $H_2$ on the same plot, so therein, we clearly see the overall sequence of boundary values, which is $2, 3, 7, 15, 18, 24, 25, 30$. Although the desired final number of buckets $\beta$ may be chosen to be any number less than or equal to 3, we drive the example merging for $\beta = 3$.

**Table** 5.1 Symbol Table Used in Section

| Symbol | Definition |
|---|---|
| $H^0$ | Pre-histogram after first assembling of $k$ exact equi-depth histograms, $H_1$, $H_2$, ..., and $H_k$ |
| $H^*$ | Final approximate equi-depth histogram after bucket merging operations of $H^0$ |
| $H^e$ | Exact equi-depth histogram for the union of $k$ value sets, $P_1$, $P_2$, ..., and $P_k$ |
| $s(i, H)$ | The $i^{th}$ bucket size of the equi-depth histogram $H$ |
| $a(i, H)$ | The $i^{th}$ approximate bucket size of the approximate equi-depth histogram $H$ |
| $S(i, H)$ | The $i^{th}$ cumulative size of the equi-depth histogram $H$ |
| $A(i, H)$ | The $i^{th}$ approximate cumulative size of the equi-depth histogram $H$ |
| $R(i, j, H)$ | Sum of bucket sizes starting from the $i^{th}$ bucket up to the $j^{th}$ bucket (both inclusive) of the equi-depth histogram $H$ |

Let us name the calculated pre-histogram (after first assembling of $H_1$ and $H_2$) as $H^0$, final merged approximate equi-depth histogram resulted from our method as $H^*$, and exact equi-depth histogram for the union of value sets $P_1$ and $P_2$ as $H^e$. Briefly, we start with assembling $H_1$ and $H_2$ in an initial pre-histogram $H^0$ and we merge consecutive buckets of $H^0$ while the merged bucket size is greater than or equal to the exact bucket size $N/\beta$ till the remaining number of buckets is equal to the desired number $\beta$. For $\beta = 3$, exact bucket sizes $N/\beta$ should be equal to $27/3 = 9$ and it can be presented as $s(1, H^e) = s(2, H^e) = s(3, H^e) = 9$. The cumulative bucket sizes for $H^e$ are $S(1, H^e) = 9$, $S(2, H^e) = 18$, and $S(3, H^e) = 27$. Now, we shall examine the creation of the pre-histogram $H^0$. The boundaries of $H^0$ are $2, 3, 7, 15, 18, 24, 25, 30$ which are shortly the sorted boundaries of $H_1$ and $H_2$. Since $H^0$ has $(T + 1) \times 2 = (3 + 1) \times 2 = 8$ boundaries, it has $8 - 1 = 7$ buckets. The important part of the creation is approximation of bucket sizes of

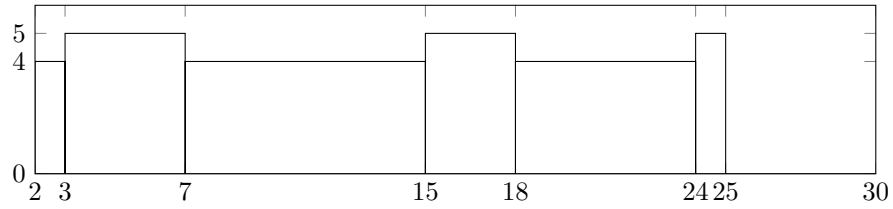
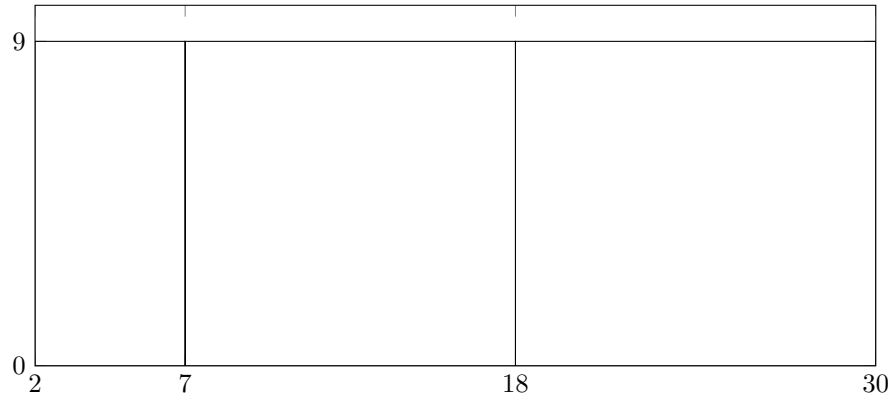



**Figure** 5.4 The initial pre-histogram $H^0$ constructed just after the first assembling of $H_1$ and $H_2$.

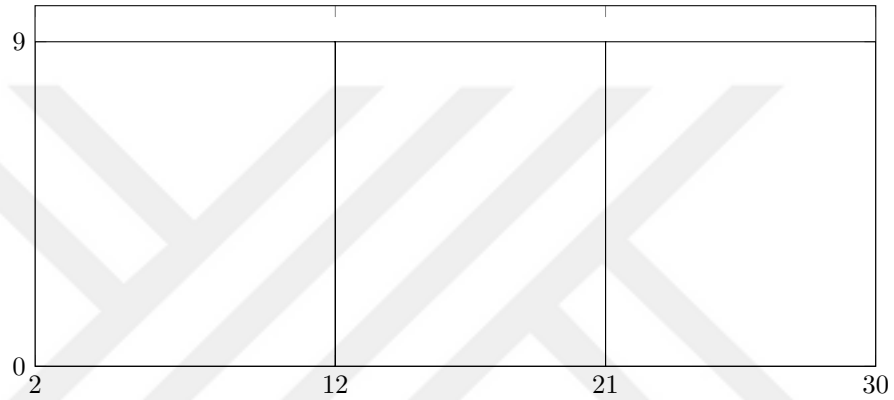
$H^0$. Before the approximation of bucket sizes, we should determine approximate cumulative bucket sizes of $H^0$ and then we are able to calculate the approximate bucket sizes from the definition of the cumulative bucket size function $A(i, H)$. The approximate cumulative bucket sizes are calculated by presuming that all values in each bucket are at the beginning boundary of the bucket. For example, let us consider the first bucket of $H_1$. In this bucket, we have values 2, 4, 5, and 6 and we suppose that all these values are at the point 2. By using this supposition, any cumulative bucket size is easily determined by summing the bucket size of the histogram which holds the next boundary and the previous cumulative bucket size starting with 0. Thus, since the first boundary of $H^0$ is 2 and this boundary is the first boundary of $H_1$, the first cumulative approximate bucket size $A(1, H^0)$ is equal to $s(1, H_1) = 4$. The second cumulative approximate bucket size $A(2, H^0)$ is equal to $s(1, H_2) + A(1, H^0) = 5 + 4 = 9$ because the next coming $H^0$ boundary is 3 and it is the first boundary of $H_2$. After the boundary 3, the next $H^0$ boundary is 7 and it is the second boundary of $H_1$. Therefore, the third cumulative approximate bucket size $A(3, H^0)$ is equal to $s(2, H_1) + A(2, H^0) = 4 + 9 = 13$. The remaining approximate bucket sizes are calculated in the same way and $A(4, H^0)$, $A(5, H^0)$, $A(6, H^0)$, and $A(7, H^0)$ are 18, 22, 27, and 27, respectively. Now, we are able to calculate approximate bucket sizes. Approximate size of the first bucket $a(1, H^0)$ relying between boundary 2 and 3 is directly equal to $A(1, H^0)$ and it is 4. The second approximate bucket size $a(2, H^0)$ is the difference between the first and the second cumulative bucket size. Thus, $a(2, H^0)$ is equal to $A(2, H^0) - A(1, H^0) = 9 - 4 = 5$. Similarly, $a(3, H^0) = A(3, H^0) - A(2, H^0) = 13 - 9 = 4$, $a(4, H^0) = 5$, $a(5, H^0) = 4$, $a(6, H^0) = 5$, and $a(7, H^0) = 0$. Graphical representation of created $H^0$ is given in Figure 5.4.

Next, we merge the buckets of $H^0$ until the remaining bucket count is equal to $\beta$. We use approximate cumulative bucket size instead of approximate bucket size to decrease the division error while merging. The merging process starts with the first bucket of $H^0$. First of all we compare the first cumulative bucket size of $H^0$, $A(1, H^0)$, with the first cumulative bucket size of exact (ideal) histogram, $S(1, H^e)$, and we see that $A(1, H^0)$ is less than $S(1, H^e)$. We continue comparing the next cumulative bucket size of $H^0$, $A(2, H^0)$, with again the first cumulative bucket size of exact (ideal) histogram, $S(1, H^e)$, and we now see that $A(2, H^0)$ is equal to $S(1, H^e)$. Again, we continue comparing. This time, we see that $A(3, H^0)$ is greater than $S(1, H^e)$.Therefore, the buckets starting from the first bucket to the third bucket except the third one (because the result of the previous comparison is equality) would be merged and this merged bucket would be the first bucket of the final merged approximate histogram, $H^*$. The resulting new bucket size would be $A(2, H^0)$ because the new merged bucket is the first bucket of $H^*$. Then, we are going to create the second bucket of $H^*$. For this creation, we continue comparing cumulative bucket sizes starting from the first not merged bucket number with the second cumulative bucket size of $H^e$. We see that $A(3, H^0)$ is less than $S(2, H^e)$. Next comparison is between the next cumulative of $H^0$ and again the second cumulative of $H^e$. This time equality is seen. We continue comparing the next cumulative of $H^0$, $A(5, H^0)$ with $S(2, H^e)$. At this point, $A(5, H^0)$ is greater than $S(2, H^e)$. Thus, we merge the buckets starting from the third one to the fifth one again except the fifth one and the created new bucket would be the second bucket of $H^*$. This merging process would end when the remaining bucket count is equal to $\beta$ and we get $H^* = \{(2, 9), (7, 9), (18, 9), (30, 0)\}$ as seen in Figure 5.5a. For comparison, $H^e$ is given in Figure 5.5b.

The generalization of this method for merging more than 2 histograms is now easy after the one given above. Let us have k value sets ($P_1$, $P_2$, ..., $P_k$) and their summaries (T-bucket equi-depth histograms, $H_1$, $H_2$, ..., $H_k$) to be merged. The merging process for the general case starts with the creation of an initial pre-histogram, $H^0$. This can be done with sorting all boundary values coming from summaries and determining approximate bucket sizes in the same way with the one described above. The calculated histogram $H^0$ has $k \times (T + 1)$ boundaries




(a) The final approximate histogram $H^*$ constructed by merging $H_1$ and $H_2$.




(b) The exact histogram for union of $P_1$ and $P_2$.

**Figure** 5.5 The final approximate and exact histograms of the example value sets $P_1$ and $P_2$.

and thus $k \times (T + 1) - 1$ buckets. The rest of the merging method is exactly the same with the case when we have only two histograms. That is, we combine consecutive buckets of $H^0$ by comparing the cumulative bucket sizes of $H^0$ with cumulative sizes of exact histogram, $H^e$, until $\beta$ buckets remain.

Algorithm 1 shows the pseudocode of the explained method above. The algorithm takes $T$-bucket equi-depth histograms of $k$ value sets, total number of values, $N$, which is the sum of all sizes of value sets, and desired bucket count of final histogram, $\beta$ as inputs and constructs and returns final approximate $\beta$-bucket equi-depth histogram, $H^*$. Lines 1 through 3 of the algorithm is performed for the creation of the initial pre-histogram, $H^0$. First, boundaries of input histograms are sorted at Line 1 and then bucket sizes are calculated according to the above example at Line 2. The subroutine CreateHistogram called at Line 3 simply

**Algorithm 1:** Equi-depth Histogram Merging

**Input**: $H_1, H_2, \ldots, H_k$ : $k$ equi-depth histograms each with $T$ buckets, $N$ : total number of values, $\beta$ : desired bucket count of final histogram

**Output**: $H^*$ : an approximate equi-depth histogram with $\beta$ buckets.

```
1  b ← {sorted boundaries of H_1, H_2, ..., H_k}
2  s ← {bucket sizes calculated as described}
3  H^0 ← CreateHistogram(b, s)
4  H* ← H^0
5  last ← 1 ; next ← 1 ; current ← 1
6  remaining ← k(T + 1) − 1
7  while remaining > β do
8      while A(next, H^0) ≤ current × N/β do
9          next ← next + 1
10     MergeBuckets(last, next − 1, H*)
11     last ← next ; current ← current + 1
12     remaining ← remaining − (next − 1 − last)
13 return H*
```

creates a histogram from given boundary and bucket size sets and at that line $H^0$ is created from $b$ and $s$. The created $H^0$ has $k \times (T + 1)$ boundaries and $k \times (T + 1) - 1$ buckets. After creation of $H^0$, it would be copied to $H^*$ at Line 4. Once $H^0$ is created and copied to $H^*$, required buckets are combined on $H^*$ considering ideal bucket size, $N/\beta$. The main `While` loop iterates until the remaining number of buckets is equal to $\beta$. The inner `While` loop given in Lines 8 and 9 seeks for the next feasible point of buckets to combine at each iteration of the main loop. When such a point is found, we apply MergeBuckets subroutine which combines buckets from $last$ to $next - 1$, both inclusive, on $H^*$ as shown in Line 10. Notice that MergeBuckets merges buckets according to the first state of bucket indexes.

For the asymptotic performance of the algorithm, sorting boundaries is likewise merging $k$ sorted lists and it can be done in $O(Tk \log k)$. Bucket sizes and CreateHistogram subroutine can both run in $O(Tk)$ at Lines 2 and 3. For the inner

loop, the increment at Line 9 can be performed at most $\beta$ times. The number of iterations for the main loop changes with the decrease in remaining bucket counts. Observe that the decrease is equal to the inner loop iteration number and MergeBuckets subroutine takes the same time with the inner loop for each main loop iteration. Considering this observation, total time required for the main loop is $O(Tk)$. Consequently, the initial sorting dominates the rest of the algorithm, and the algorithm runs in $O(Tk \log k)$-time.

Let us debug the algorithm line by line for the two example 3-bucket equi-depth histograms $H_1$ and $H_2$ given in Figures 5.1 and 5.2. Recall that $H_1$ and $H_2$ are histograms of value sets $P_1$ and $P_2$. Therefore $N$ is equal to $|P_1|+|P_2| = 12+15 = 27$. Let $\beta$ is equal to 3. We know $H^0 = \{(2,4),(3,5),(7,4),(15,5),(18,4),(24,5),(25,0),(30,0)\}$ from the given detailed explanation above. In addition, the start state of $H^*$ is the same as $H^0$. The variables *last*, *next*, and *current* is equal to 1 and *remaining* is calculated as $k(T+1)-1 = 2(3+1)-1 = 7$. Because the *remaining* is greater than $\beta$ at current state, we enter the main loop. For the inner loop, $A(1,H^0)$ and $A(2,H^0)$ is less or equal to $current \times N/\beta$ which is $1 \times 27/3 = 9$ but $A(3,H^0)$ is greater than 9. Hereby, inner `While` loop 2 times and *next* would be 3. Then, MergeBuckets subroutine merges the buckets 1 and 2 of $H^*$. The illustration of $H^*$ is shown in Figure 5.6 after merging. The variables *last* and *current* are updated after the execution of MergeBuckets is finished and *last* would be 3 and *current* would be 2. The *remaining* variable, keeping the remaining bucket number of $H^*$, would be 6 after the calculation is done at Line 12. The main loop finishes after $H^*$ has $\beta$ buckets and execution of the algorithm ends with returning the created $H^*$.

We also show another example including all steps of algorithm in figures from Figure 5.7 to Figure 5.14.

Now, we discuss the error bounds of the output histogram $H^*$. The following two theorems and their proofs verify the error bounds on bucket sizes and the sum of any range of bucket sizes of $H^*$.

**Theorem 1.** *Let $H_1$, $H_2$, ..., $H_k$ be $T$-bucket equi-depth histograms of value sets $P_1$, $P_2$, ..., $P_k$, and $H^*$ be the approximate $\beta$-bucket equi-depth histogram*

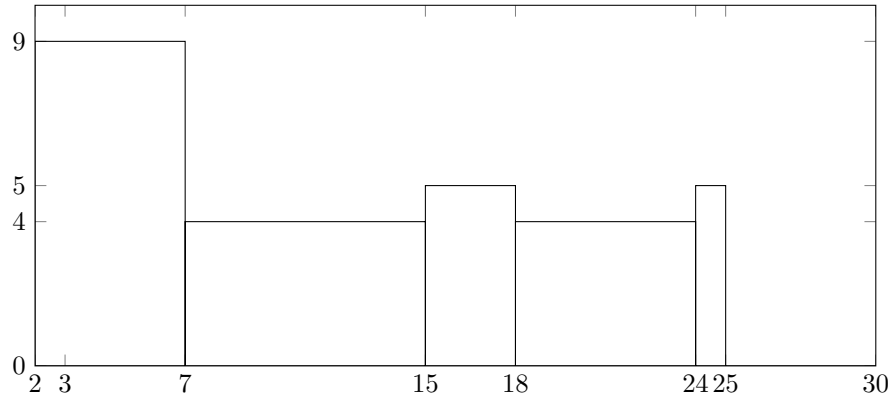



**Figure** 5.6 The state of $H^*$ after the first iteration of main loop of Algorithm 1.


**Figure** 5.7 A sample equi-depth histogram $H$ with 3 buckets, based on data {2,4,5,6,7,10,13,16,18,20,21}. When considered sorted sequence of the underlying data, first bucket contains first three ({2,4,5}), second bucket contains four ({6,7,10,13}), and third (also the last) bucket contains last four ({16,18,20,21}). Then, $s(1,H) = 3$ and $s(2,H) = s(3,H) = 4$. For the cumulative sizes, $S(1,H) = 3$, $S(2,H) = 7$, and $S(3,H) = 11$.

*where $\beta \leq T$ constructed by the algorithm. Then, the size of any bucket $a(i,H^*)$ is $(N/\beta) \pm \varepsilon_{max}$ where $\varepsilon_{max} < 2\beta/T \times (N/\beta)$.*

*Démonstration.* Recall that the calculations of bucket sizes of $H^0$ depends on supposition that all values in each bucket are at the beginning boundary of the bucket and $H^*$ is some-buckets-merged version of $H^0$. Now consider an $i^{th}$ bucket between the $i^{th}$ and the $i+1^{th}$ boundaries (boundaries may be any of the two consecutive boundaries of $H^0$) of $H^*$ illustrated in Figure 5.15. As seen in the figure, all of the values in the buckets divided by the $i^{th}$ boundary may stay at the right hand side of the boundary in contrast to our assumption and all values in the buckets divided by the $i+1^{th}$ boundary may stay at the left hand side of

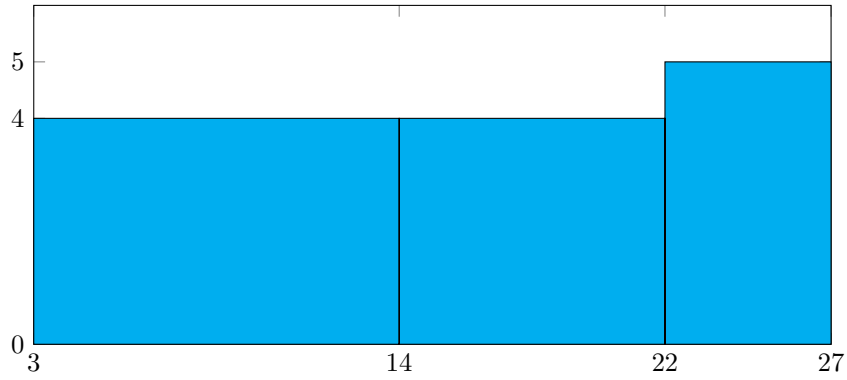



**Figure** 5.8 Another sample equi-depth histogram $H$ with 3 buckets, which represents data {3,9,11,12,14,15,17,19,22,23,24,26,27} of size 13. First bucket has five ({3,9,11,12}), second bucket contains five ({14,15,17,19}), and last bucket has seven ({22,23,24,26,27}) of them. Then, $s(1, H) = s(2, H) = 4$, and $s(3, H) = 5$. For the cumulatives, $S(1, H) = 4$, $S(2, H) = 8$, and $S(3, H) = 13$.

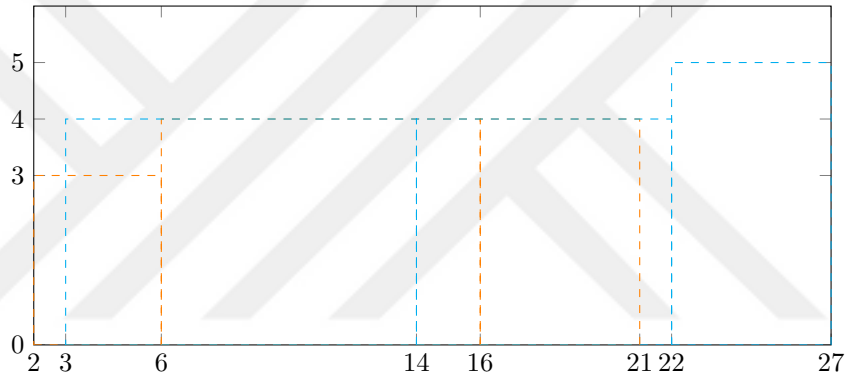



**Figure** 5.9 Sample equi-depth histograms given in Figures 5.7 (orange) and 5.8 (cyan) are coupled together, with boundary sequence 2, 3, 6, 14, 16, 21, 22, 27.

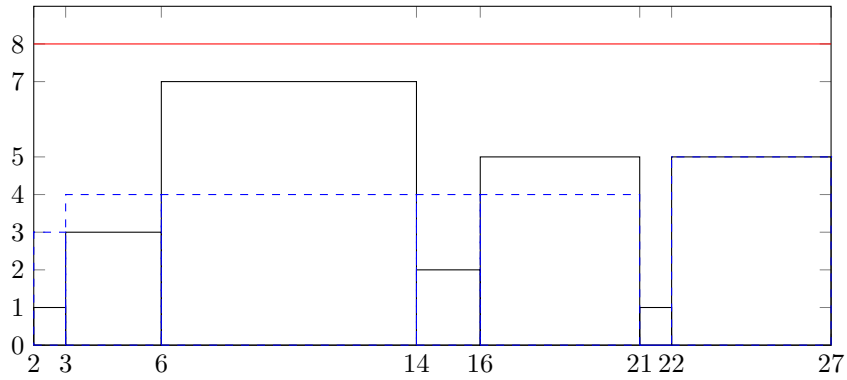



**Figure** 5.10 The initial histogram $H^0$ (black solid line) representing data {2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27} with $T^0 = 7$ buckets. Approximate bucket sizes (blue dashed line) with respect to $H^0$. The ideal size of a bucket (red straight line at 8) for an equi-depth histogram.

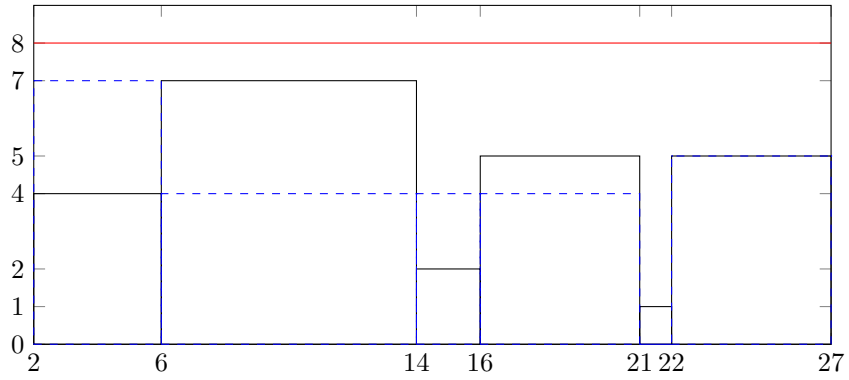



**Figure** 5.11 The histogram $H^1$ (black solid line), which is obtained by merging first two buckets of $H^0$, thus containing $T^1 = 6$ buckets. Approximate bucket sizes (blue dashed line) with respect to $H^1$. The ideal size of a bucket (red straight line at 8) for an equi-depth histogram.

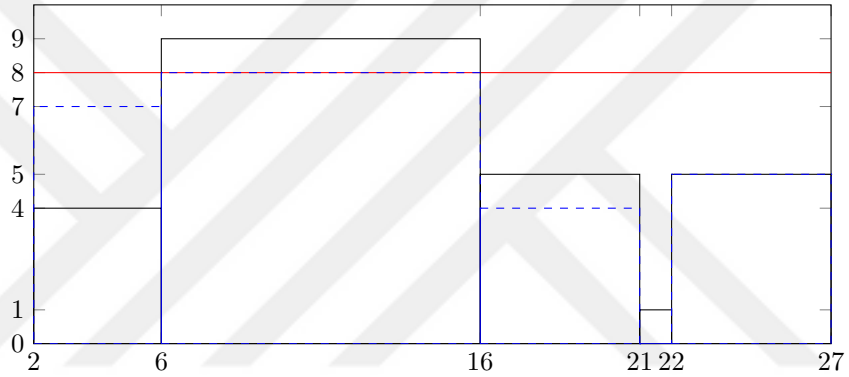



**Figure** 5.12 The histogram $H^2$ (black solid line), which is obtained by merging second and third buckets of $H^1$, thus containing $T^2 = 5$ buckets. Approximate bucket sizes (blue dashed line) with respect to $H^2$. The ideal size of a bucket (red straight line at 8) for an equi-depth histogram.

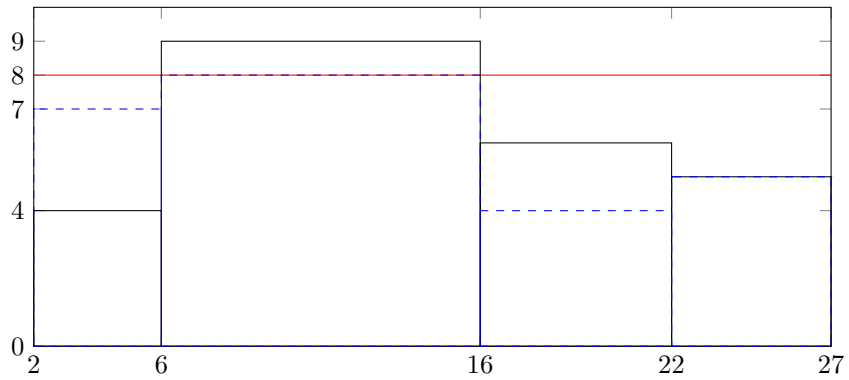



**Figure** 5.13 The histogram $H^3$ (black solid line), which is obtained by merging third and fourth buckets of $H^2$, thus containing $T^3 = 4$ buckets. Approximate bucket sizes (blue dashed line) with respect to $H^3$. The ideal size of a bucket (red straight line at 8) for an equi-depth histogram.

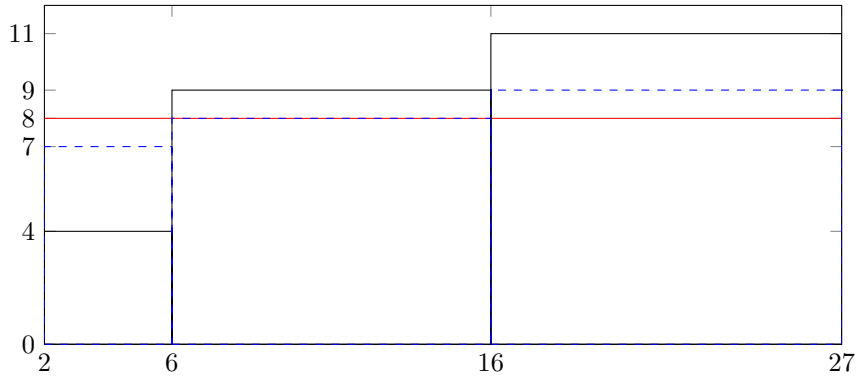



**Figure** 5.14 The histogram $H^4$ (black solid line), which is obtained by merging third and fourth buckets of $H^3$, thus containing $T^4 = 3 = B$ buckets. Approximate bucket sizes (blue dashed line) with respect to $H^4$. The ideal size of a bucket (red straight line at 8) for an equi-depth histogram.

the boundary. In this case, $a(i, H^*)$ gets the maximum value. Vice versa, $a(i, H^*)$ gets the minimum value in the case that all possible values in the divided buckets stay out of the $i^{th}$ bucket in contrast to the case seen in Figure 5.15. The following calculation shows the maximum value of $a(i, H^*)$.

$$\begin{aligned} a(i, H^*)_{max} &= C + |P_1|/T + |P_3|/T \\ &\quad + \cdots + |P_k|/T \\ &\quad + |P_1|/T + |P_2|/T \\ &\quad + \cdots + |P_{k-1}|/T \end{aligned} \tag{5.5}$$

where $C$ is constant which is the sum of the sizes of the buckets relying completely in the $i^{th}$ bucket and $|P_1|, |P_2|, \ldots, |P_k|$ is the size of sets. Adding and subtracting

$|P_2|/T$ and $|P_k|/T$ to the equation, we get the following equation.

$$\begin{aligned} a(i,H^*)_{max} &= C + (|P_1| + |P_2| + \cdots + |P_k|)/T \\ &\quad + (|P_1| + |P_2| + \cdots + |P_k|)/T \\ &\quad - |P_2|/T - |P_k|/T \\ &= C + 2N/T - |P_2|/T - |P_k|/T \\ &< C + 2N/T \end{aligned} \tag{5.6}$$

And $a(i,H^*)_{min}$ is equal to $C$ because no additional values are located in the $i^{th}$ bucket except the constant ones. Once $a(i,H^*)_{max}$ and $a(i,H^*)_{min}$ are determined, $\varepsilon_{max}$ would be the difference between them.

$$\begin{aligned} \varepsilon_{max} &= a(i,H^*)_{max} - a(i,H^*)_{min} \\ &< C + 2N/T - C \\ &< 2N/T \end{aligned} \tag{5.7}$$

The following equation shows another expression of $\varepsilon_{max}$ in terms of exact (ideal) bucket size $N/\beta$.

$$\begin{aligned} \varepsilon_{max} &= 2N/T \\ &< 2N\beta/T\beta \\ &< 2\beta/T \times (N/\beta) \end{aligned} \tag{5.8}$$

■

**Theorem 2.** *Let $H_1$, $H_2$, ..., $H_k$ be $T$-bucket equi-depth histograms of value sets $P_1$, $P_2$, ..., $P_k$, and $H^*$ be the approximate $\beta$-bucket equi-depth histogram where $\beta \leq T$ constructed by the algorithm. Then, the size of any range spanning $m$ buckets, $R_a(i, i+m, H^*)$, is $m \times (N/\beta) \pm \varepsilon_{max}$ where $\varepsilon_{max} < 2\beta/T \times (N/\beta)$.*

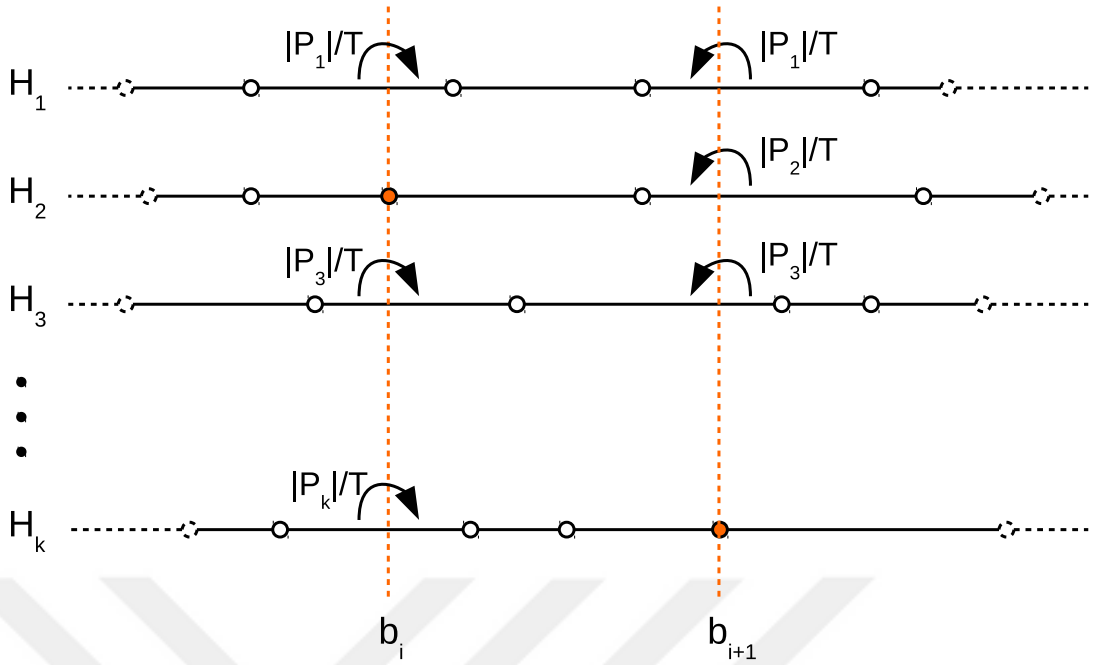



**Figure** 5.15 Illustration of maximum bucket size.

*Démonstration.* Let us start with proving the error bound of range size of two consecutive buckets. Figure 5.16 shows this case. There are two consecutive buckets and three boundaries ($b_i$, $b_{i+1}$, and $b_{i+2}$), the middle one ($b_{i+1}$) splits the two buckets. Notice that the intersected buckets by $b_{i+1}$ completely rely in the range of the two buckets and this means that the sizes of these buckets are added as a constant to the range size $R_a(i, i+1, H^*)$. As a result, this proof turns into the proof of error bound of bucket size given in Proof 13 and Equation 5.7 and Equation 5.7 also proves Theorem 2. The general case -spanning ranges includes more than two buckets- can also transform into a single bucket problem in the same way with the case with two buckets. ■

According to Theorems 1 and 2, users can bound the maximum bucket size error of final $\beta$-bucket approximate equi-depth histogram $H^*$ by selecting appropriate bucket numbers $\beta$ and $T$. For example, let us calculate $T$, number of buckets of equi-depth histograms of data partitions kept in the summary files, in terms of $\beta$ for getting final merged histograms, the maximum bucket size errors of which do not exceed 5% of the ideal bucket size ($N/\beta$). If we use Equation 5.8, we can find the minimum number of buckets $T$ needed to satisfy the 5% error condition

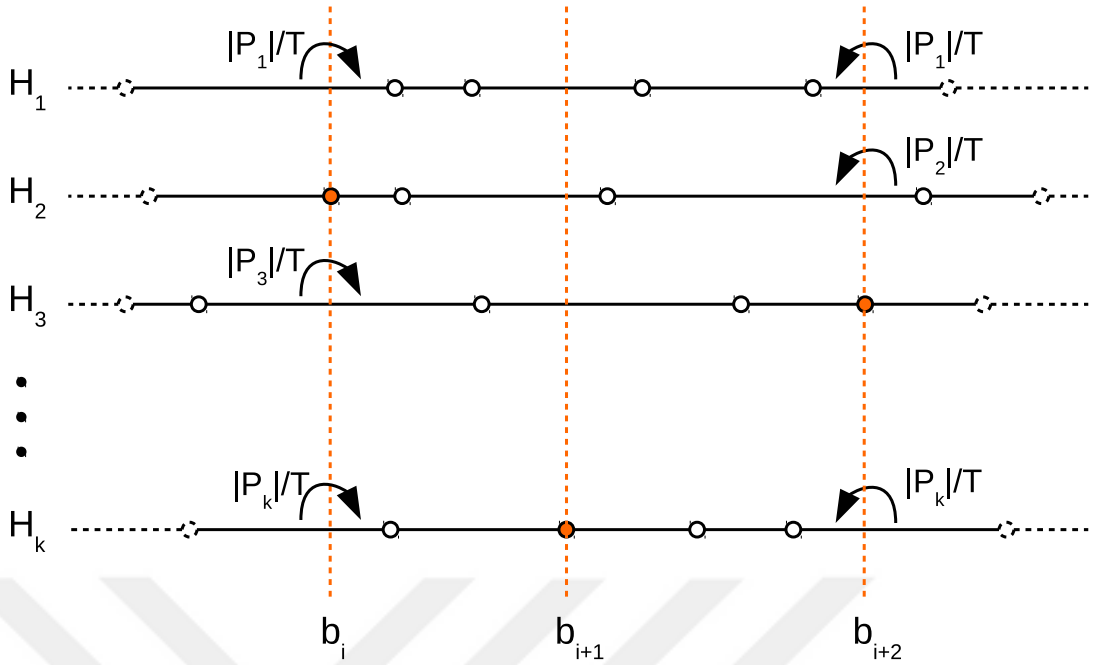



**Figure** 5.16 Illustration of maximum size of a range of buckets.

as follows.

$$\varepsilon_{max} < 2\beta/T \times (N/\beta) \leq 0.05(N/\beta)$$

$$40\beta \leq T$$

Consequently, the required bucket size $T$ should be at least 40 times $\beta$ which is the desired number of buckets of constructed histograms using our method.

# CHAPTER 6

## IMPLEMENTATION WITH HADOOP MAP-REDUCE

In this section, we explain the implementation details of our histogram processing framework on Hadoop MapReduce. The framework consists of two main MapReduce jobs. One of them is named as Summarizer which runs offline and is scheduled for summarizing the new coming data to HDFS. The Summarizer constructs a $T$-bucket equi-depth histogram of the data. After summarizing, the resulting equi-depth histograms are stored in HDFS. The architecture of Summarizer is displayed in Figure 6.1. The second job, Merger, is run on-demand according to users' requests. Its duty is to merge the related summaries from HDFS by considering user requests and to construct the final $\beta$-bucket approximate equi-depth histogram. Its architecture is showed in Figure 6.2.

The overview picture of the histogram processing framework is given in Figure 6.3. In the left of the figure, HDFS holds whole data including the new data, summary files, and created histograms according to user requests. The framework is in the right of the picture and Summarizer and Merger jobs take place in the framework. Every time, new data is pushed to HDFS, the Summarizer constructs its summary ($T$-bucket equi-depth histogram) and saves it to HDFS, again. When a user requests an equi-depth histogram of desired partitions (it can be any set of data partitions), the Merger processes the request by merging the related summaries of desired partitions and saves the merged final histogram to HDFS. These jobs can also be implemented in the Hive and Pig as user functions.

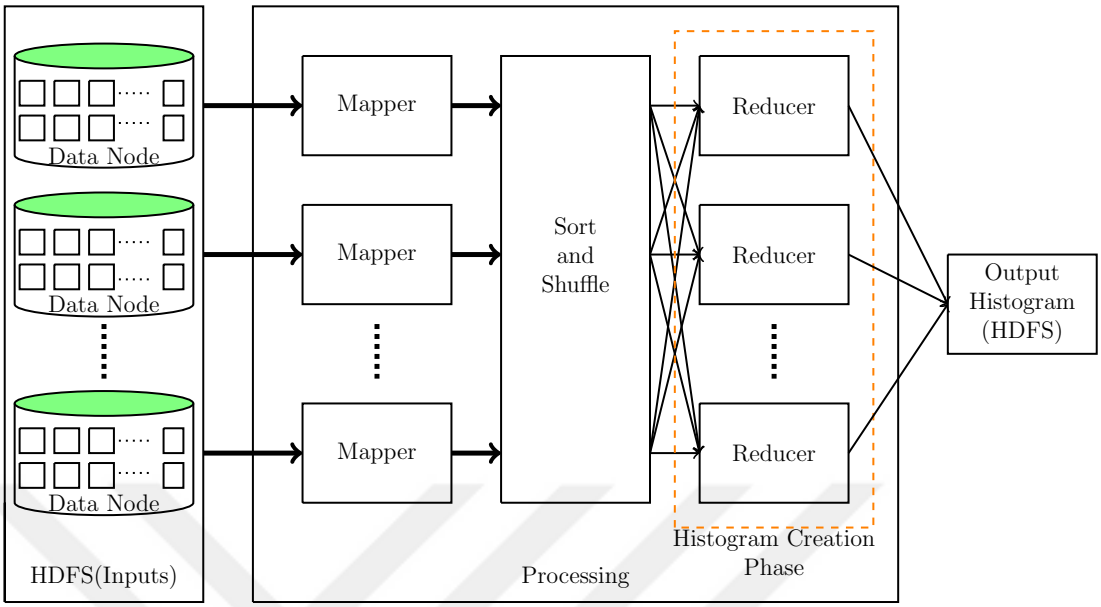



**Figure** 6.1 Architecture of Summarizer.The right of the figure, Histogram Creation Phase is demonstrated with dashed lines.

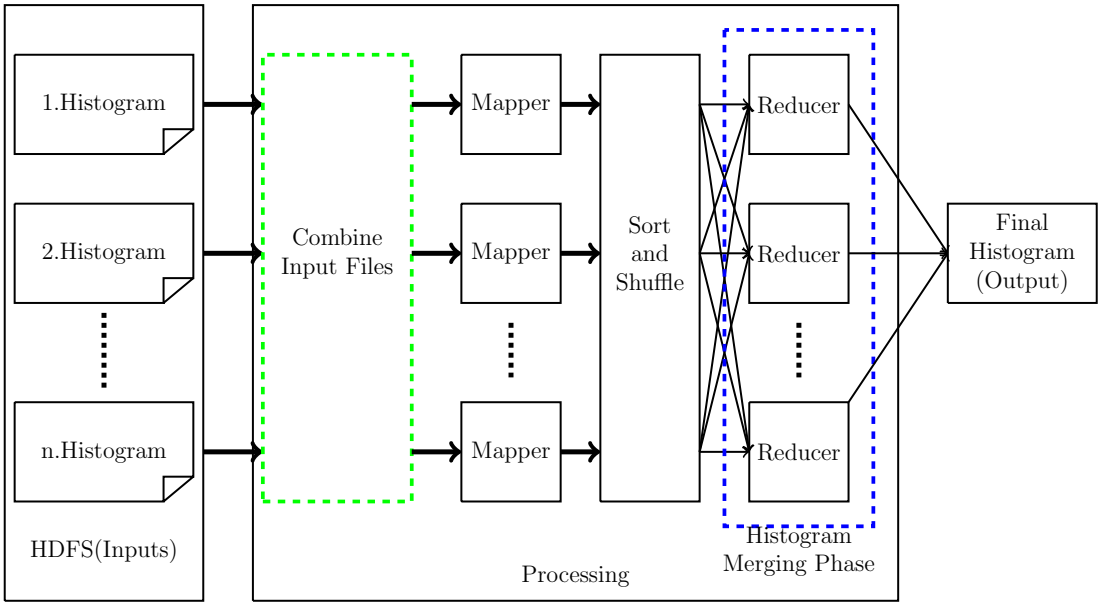



**Figure** 6.2 Architecture of Merger. Histogram Merging Phase showed in the right of the figure gets sorted values of summaries and merges.

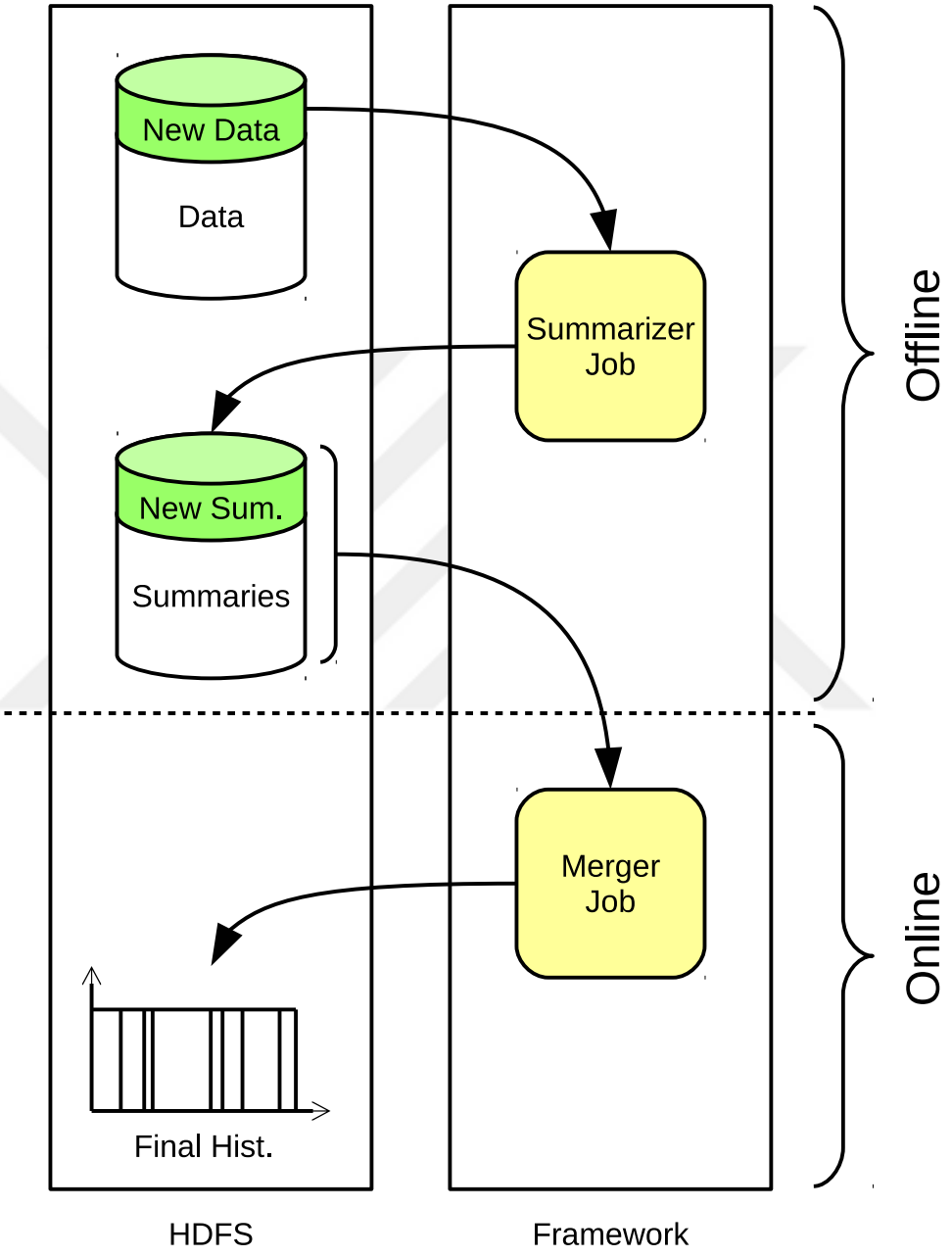



**Figure** 6.3 Overview of Proposed Framework.

# CHAPTER 7

## EXPERIMENTAL RESULTS

In this section, we will describe how we tested the proposed method. The method was implemented on Hadoop MapReduce framework and was tested on two different datasets. One of them is synthetic data with 155 million of tuples created by using Gumbel distribution for skewness to represent the response of the method for skewed data. The other one is 295 GB uncompressed real data which is taken from hourly page view statistics of Wikipedia. The data consists of approximately 5 billion tuples which belong to January 2015 and each tuple has 4 columns which are *language*, *pagename*, *pageviews* and *pagesize*. We used *pagesize* for histogram construction. The proposed method (*merge*) is compared with corrected tuple level random sampling (*tuple*). By definition, bare tuple level random sampling collects tuples randomly and constructs histogram with collected tuples but doing so does not work well when the data is sparse at the edges. Therefore, we fix this problem by including the edge values to the collected tuples by default. As mentioned in Section 3, histogram construction of the data coming from daily logs is an important issue and tuple level random sampling method is also unfavorable for constructing a histogram of a given time interval. Hereby, sampling stage of tuple level is run offline to compare time spendings.

All equi-depth histograms are build with bucket size of 254 as used by Oracle as default bucket size for histogram enhancement. We fundamentally run two types of tests to represent the effectiveness of the proposed method in terms of boundary and bucket size error and run time. The first test represents the results according to T changes which is daily exact histogram bucket size for the proposed method and sample size for tuple level sampling. The second test represents the run time efficiency of histogram construction for the changes in a given time interval.

Approximate histograms may have two types of error. One of them is that approximate histograms may not have the same bucket boundaries with the exact ones and the other is that bucket sizes of the approximate histograms may deviate from the exact ones. The former error is named as boundary error ($\mu_b$) and

defined as follows :

$$\mu_b = \frac{B}{v_{max} - v_{min}} \sqrt{\frac{1}{B+1} \sum_{i=1}^{B+1} [b(i, H^*) - b(i, H)]^2}. \tag{7.1}$$

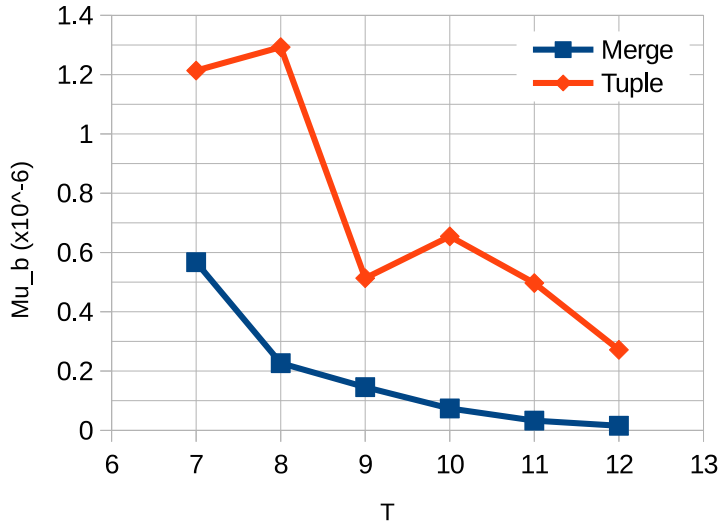
where $B$ is the bucket size, $v_{max}$ and $v_{min}$ are maximum and minimum values in relation $R$ respectively, and the function $b(i, H)$ is the $i^th$ value of a given histogram $H$. $\mu_b$ is the standard deviation of boundary errors normalized with respect to the mean boundary length $(v_{max} - v_{min})/B$. The latter error is named as size error ($\mu_s$) and formulated as follows :

$$\mu_s = \frac{B}{N} \sqrt{\frac{1}{B} \sum_{i=1}^{B} [s(i, H^*) - s(i, H)]^2}. \tag{7.2}$$

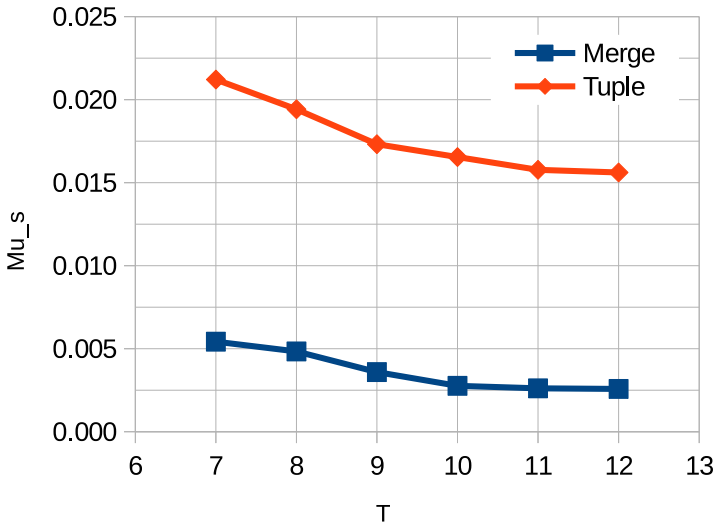
where $N$ is the total number of elements in relation $R$ and function $s(i, H)$ is the size of the $i^th$ bucket of a given histogram $H$. $s(i, H)$ is equal to the mean bucket size $N/B$ for all $i$ values in the range of 1 to $B$ if the given histogram $H$ is an equi-depth histogram. $\mu_s$ is the standard deviation of bucket size errors normalized with respect to the mean bucket size $N/B$.

## 7.1 Effect of $T$

Changing $T$ value effects the accuracy of the constructed approximate equi-depth histogram. The first experiment is run to show the effects of $T$ changes on both the proposed method and tuple level sampling. Figures 7.1 and 7.2 show the error graphs of the approximate equi-depth histograms constructed using the proposed method and tuple level sampling method. According to the graph in Figure 7.1a, the constructed histogram by using *merge* method for real data is at least 2 times more accurate in terms of boundary error $\mu_b$ than the one constructed using *tuple* method. Moreover, the $\mu_b$ error for *tuple* method is not consistent because of the randomness and the construction process must be repeated for consistency and this is not convenient because of the run time. For example, let us consider the graph of $\mu_b$ against T given in Figure 7.1a. Notice that the $\mu_b$ error for *tuple* method is not consistent. The expected result is that $\mu_b$ should decrease while $T$ increases. On the other hand, it is clearly seen from the graphs of *merge* method in Figures 7.1a, 7.1b, 7.2a, and 7.2b that $\mu_b$ is a non-decreasing function of $T$. The reason of this consistency is the maximum error bound of *merge* method



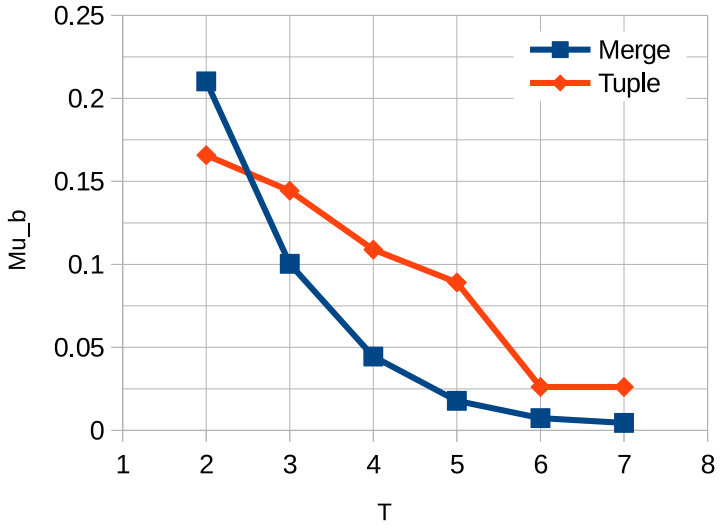

(a) Graph of $\mu_b$ against $T$ for real data



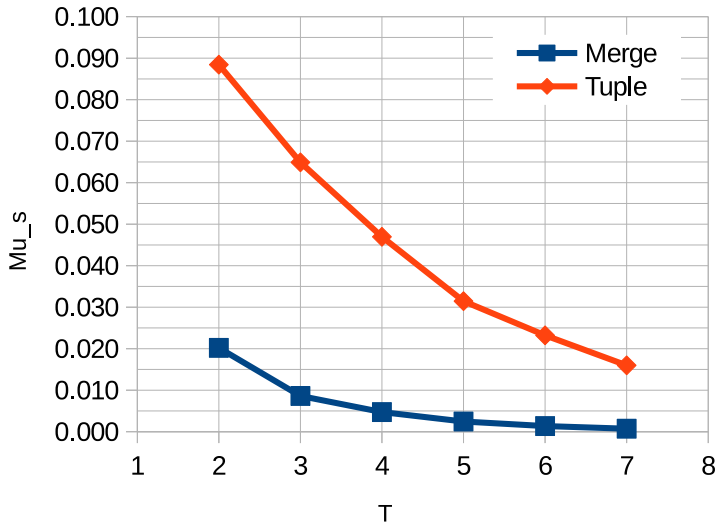

(b) Graph of $\mu_s$ against $T$ for real data

**Figure** 7.1 Lin-log graphs of error metrics against $T$ ($B \times 254 \times 2^n$) which is summary size in *merge* method and sampling size in *tuple* for real data

described in previous sections in detail. The mean running times for all methods are given in Table 7.1. Run times for merging daily summaries and samplings are nearly the same. But required time for the summarization stage of *merge* method is more than the time for offline sampling stage of *tuple* method. The reason of this time difference is that summarization is exact histogram construction and exact histogram construction for each data partition requires a complete MapReduce Job with a mapper and a reducer and the data comes from each mapper subjected



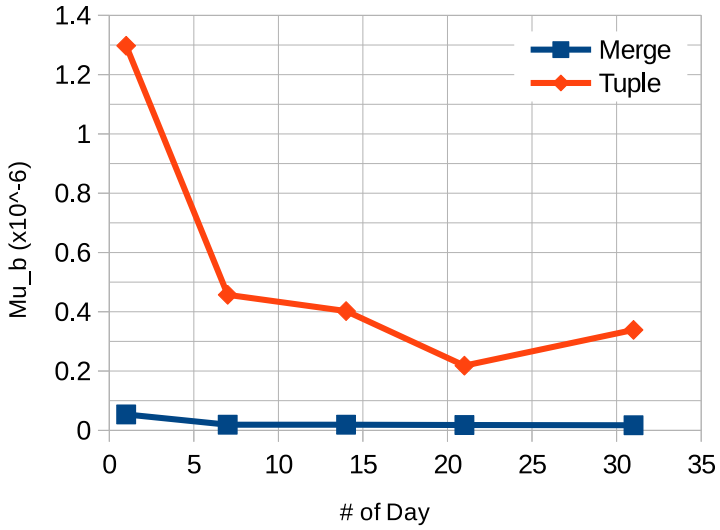

(a) Graph of $\mu_b$ against $T$ for skewed data



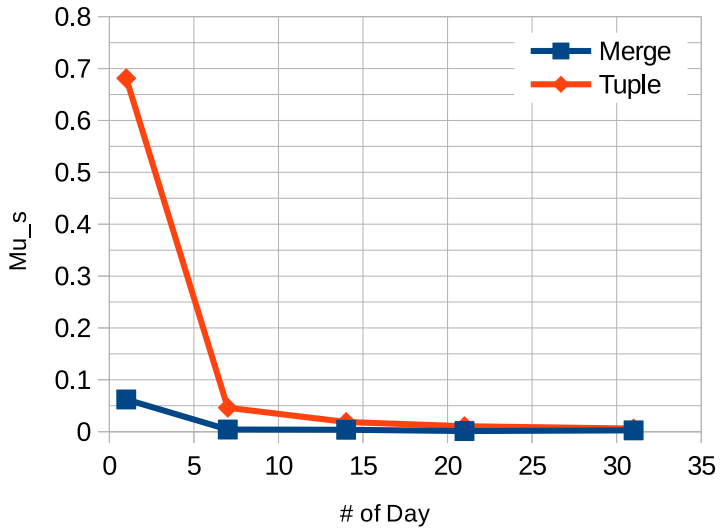

(b) Graph of $\mu_s$ against $T$ for skewed data

**Figure** 7.2 Lin-log graphs of error metrics against $T$ ($B \times 254 \times 2^n$) which is summary size in *merge* method and sampling size in *tuple* for skewed data

to shuffle and sort phase. On the other hand, tuple level random sampling does not require a reducer because the randomly selected tuples would be stored directly without sorting. Because of this difference, summarization of each day for real data takes approximately 12 minutes while sampling takes 4 minutes. Although this time efficiency of *tuple* method, all the daily summarizations and samplings are done offline, it makes *merge* method convenient for real life applications.




(a) Graph of $\mu_b$ against merged # of days for real data

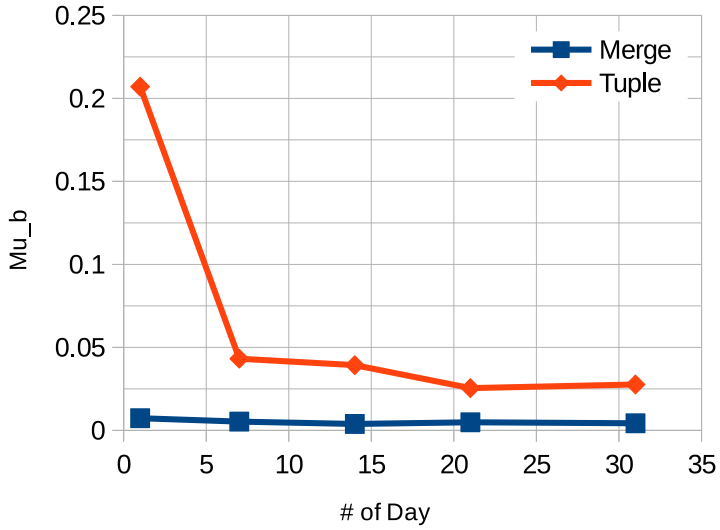



(b) Graph of $\mu_s$ against merged # of days for real data

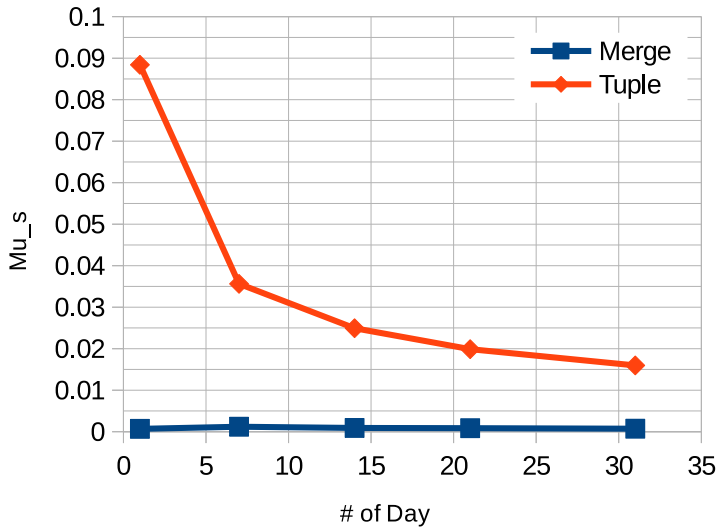
**Figure** 7.3 Graphs of error metrics against merged # of days for real data

## 7.2 Effect of given time interval

Histogram constructions according to the given time interval effect deviation of approximate histograms from the exact ones and running times. We compared *merge* method with *tuple* method with an experiment for 5 different time intervals (1 day, 1 week, 2 weeks, 3 weeks, and 1 month). $T$ value is taken to be $B \times 254 \times 2^{12}$ for real data and to be $B \times 254 \times 2^{7}$ for skewed data. All daily exact histogram and samplings are computed offline. In Figures 7.3 and 7.4, graphs of



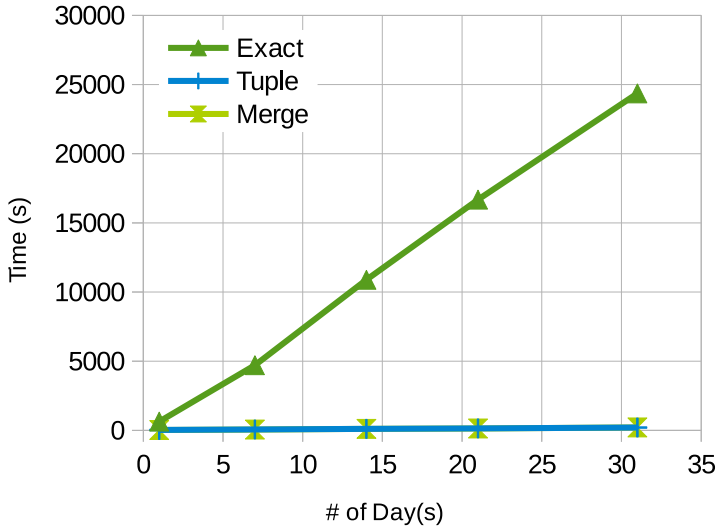

(a) Graph of $\mu_b$ against merged # of days for skewed data



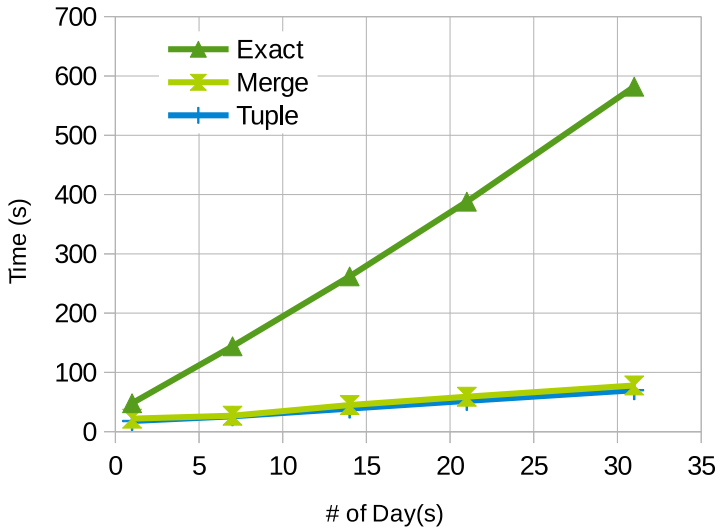

(b) Graph of $\mu_s$ against merged # of days for skewed data

**Figure** 7.4 Graphs of error metrics against merged # of days for skewed data

error metrics against time intervals are given and it is clearly seen from the graphs that the proposed method produces more sensible histograms than the ones produced using tuple level random sampling. In particular, a real data histogram constructed using *merge* method has at least 10 times less boundary error than the one constructed using *tuple* method in Figure 7.3a. Besides, again the consistency is an issue for *tuple* method as seen in Figures 7.3a and 7.4a. The graphs of running time against number of days are given in Figure 7.5. More specifically, the compared methods (*merge* and *tuple*) run in nearly the same time duration




(a) Running time against merged # of days for real data




(b) Running time against merged # of days for skewed data

**Figure** 7.5 Graphs of running time against merged # of days

except offline parts.

**Table** 7.1 Mean Running Times of Monthly Equi-depth Histogram Construction

| Method | Real Data | Skewed Data |
|---|---|---|
| Exact Hist. Construction | 24358 | 582 |
| Tuple with Online Sampling | 6169 | 68 |
| Summarizing for Each Day | 725 | 18 |
| Merging of Daily Summaries | 117 | 73 |
| Sampling for Each Day | 223 | 18 |
| Merging of Daily Samplings | 113 | 71 |

# CHAPTER 8

# DEMONSTRATIONS

## 8.1 Overview of Demonstration

In this section, we will explain our demonstration with implementation details. We have two demonstrations. One of them consists of two Pig Latin scripts which are called SummarizerPig and MergerPig, respectively. SummarizerPig runs offline and is used to construct exact equi-depth histogram of new comming data to HDFS with pre-specified number of buckets by user. Output of the SummarizerPig which can be called *summary* is also stored in HDFS. The other script, HistogramMerger runs online with desired query request by user. It merges SummarizerPig's desired outputs considering specified query and builds final approximate equi-depth histogram. The second demonstration is used for same goals as HistogramMerger, but there are minor differences. We call this demo as MergerWeb because a Map-Reduce job works background of JSF web page. In the MergerWeb, user prepare a query and send to the Map-Reduce job. The job gets inputs which are summaries from HDFS according to query and builds approximate equi-depth histogram as MergerPig.

In Figure 8.1, an Overview of demonstrations processing is given. The online and offline parts can be seen in the right side of the figure. Raw data, summary files and final approximate histograms constructed according to user demands stored in HDFS is displayed in the left of the figure. When new data pushed to HDFS, SummarizerPig works, constructs its summary and stores in HDFS, again. It is remembered that constructed $T$-buckets equi-depth histogram is exact histogram. When a user requests an equi-depth histogram of desired data partitions, MergerPig gets summaries of related data partitions, computes approximates $\beta$-buckets equi-depth histogram and saves output which is final merged histogram to HDFS. In this research, we have two Hadoop Map-Reduce jobs alternatives of SummarizerPig and MergerPig. The first job can be used instead of SummarizerPig works offline. The second job which is alternative of MergerPig works

Listing 8.1: Script of SummarizerPig

```
register HistogramSummarize.jar;
define Hist HistogramSummarize( 254 );
A = load  pagestat/20160101 using PigStorage(  ) as (lang:chararray,
    url:chararray,req:long,byte:long);
B = foreach A generate byte as (byte:long);
C = group B all;
D = foreach C
sorted = order B by byte;
generate Hist(sorted);
 ;
store D into  Summaries/20160101 ;
```

on-demands. Our demo is a web application gets query from user and runs Map-Reduce jobs with this query. As a result, it merges exact histograms and builds approximate equi-depth histogram.

## 8.2 Demonstrations Examples

We now present our so simple applications in this section. We have described jobs of demonstrations in previous section. In Listing 8.1 and Listing 8.2, you can see our first demo scripts. We write 'HistogramSummarizer' user define function and register it in line 1 of Listing 8.1 to create exact equi-depth histogram. The number of buckets $T$ is defined in line 2 of Listing 8.1. In the other part of script data is loaded, histogram is generated and daily histogram stored in HDFS under 'Summaries' directory.

The MergerPig 8.2 merges desired daily exact-equi-depth histograms. To use this method in Apache Pig, we write another user define function registered in 1 of Listing 8.2. As you can see in this script, the number of buckets of approximate histogram is defined in line 2 of Listing 8.2. The merged histogram data of desired time interval(time interval is a week in following example) is loaded in line 3 of following listing. In the rest of script, approximate equi-depth histogram is built and stored in HDFS.

We implement two Hadoop Map-Reduce jobs for our second demonstration.

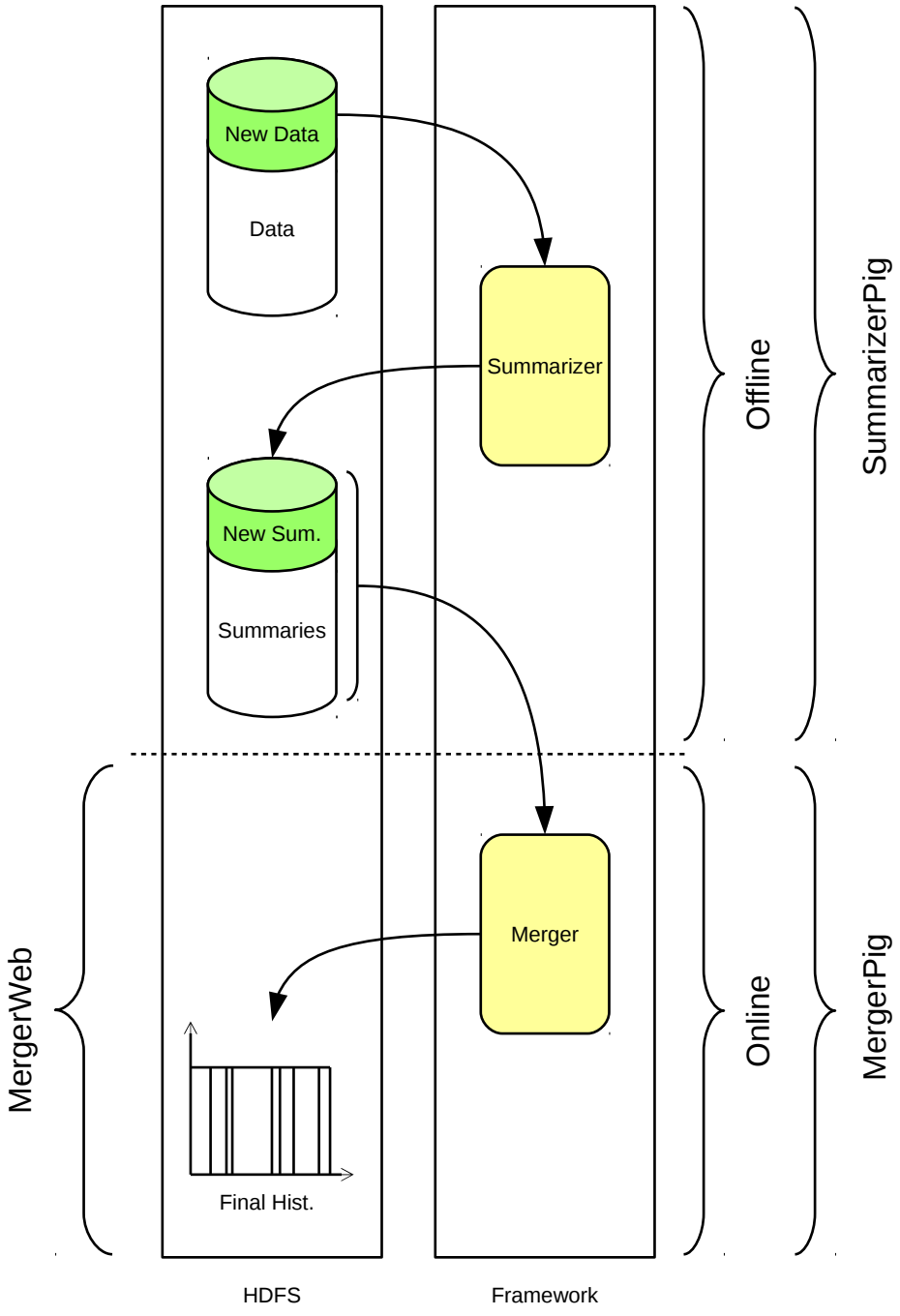



**Figure** 8.1 Overview of Demonstrations

Listing 8.2: Script of MergerPig

```
register HistogramEstimate.jar;
define Hist HistogramEstimate( 254 );
A = load  Summaries/2016010[1 7]/  as (B: bag  T: tuple(bound:long,
    numberOfTuple:long) );
B = foreach A generate flatten(B);
C = group B all;
D = foreach C
sorted = order B by bound;
generate Hist(sorted);
 ;
store D into output;
```

The first job constructs exact equi-depth histogram as 'SummarizerPig', but we doesn't present it in Figure 8.1. Because 'SummarizerPig' is enough to present working logic in it. The second job works as 'MergerPig'. Essential command to run this job is displayed in following listing.

Listing 8.3: Essential Command to Run MergerJob

```
hadoop jar HistogramEstimate.jar edu.tou.HistogramEstimate
    numberOfBuckets(  beta ) startdate enddate Summaries output
```

Actually, we only use this job in *MergerWeb* which is written in JSF and Maven frameworks. The user interface is available in Figure 8.2. In order to use this demo, input fields, which are number of buckets($\beta$), start date, end date, input files path and output file path must be filled and clicked button. As soon as job is finished, the constructed approximate equi-depth histogram of time interval is demonstrated as Figure 8.3. If you look carefully to Figure 8.3, you will notice visual disturbances and unnecessary last bucket. The visual disturbance stems from Primefaces chart component we have used. If a suitable visual component to equi-depth histogram, the histogram obtains perfectly.

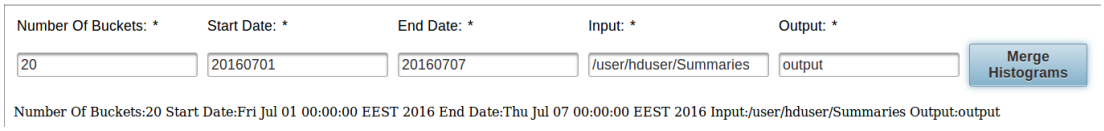



**Figure** 8.2 MergerWeb Form

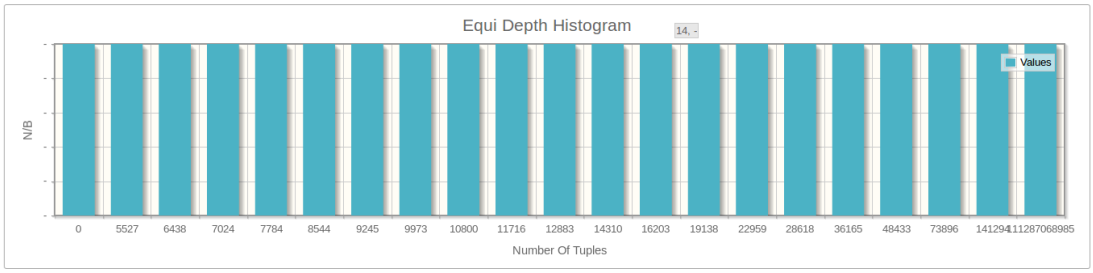



**Figure** 8.3 Equi-Depth Histogram Constructed by MergerWeb

## 8.3 Performance Comparison of Demonstrations

We will share and describe demonstrations of proposed method in this section. If we remember from previous sections, The proposed method consists of offline and online parts. For offline parts, we run HistoramSummarizer which is Map-Reduce job and SummarizerPig 8.1 and compare them in terms of time. In the crucial part of proposed method, in the online part, MergerWeb 8.2 and MergerPig 8.2 runs. We also compare its results in terms of time. The demonstrations tested on real datasets taken from hourly page view statistics of Wikipedia which are 3 GB uncompressed and consist of 35 million tuples which belongs to first hour of first week days of July 2016. The characteristic of data is widely described in Experimental Results section. All histograms are created with 254 buckets size as previous tests and oracle does. When doing these tests, a hadoop cluster consists of two node is used. The master node is Toshiba Satellite PC which has Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz and 4 GB RAM and the slave node is Raspberry Pi 3.

HistogramSummarizer job creates daily equi-depth histogram faster than SummarizerPig as shown in Figure 8.4. The sum of avarage map and shuffle times of SummarizerPig and HistogramSummarizer is nearly same. It is demonstrated comparison results in Figure 8.5. However, the spent time in reduce phase of SummarizerPig is more. Therefore, implemented pig script to construct daily

equi-depth histogram runs slower than Map-Reduce job(see Figure 8.6).

We also compared MergerWeb and MergerPig in terms of total running time and phases times as map time, shuffle time and and reduce time. We created approximate equi-depth histograms for first 2,3,5 and 7 days, respectively and compared its results. According to Figure 8.7, when MergerWeb runs, as input data grows, namely as number of summaries to be merged increase, histogram merging time increases. MergerPig optimize maps(see Figure 8.8) and runs nearly same times with minimal increases. The time spent for shuffle phase may change because of order of the data. In Figure 8.9, the time spent for shuffle phase is displayed. However, the reduce phase of MergerWeb runs faster than MergerPig as showed in Figure 8.10.

## 8.4 Conclusion of Demonstrations

We experience Map-Reduce jobs run faster than Apache Pig scripts. This situation did not surprise us because Hadoop Map-Reduce give full control to developers. But Apache Pig transforms query to Map-Reduce job series. Therefore, because of transformations and more Map-Reduce jobs, the good performance of Hadoop Map-Reduce jobs is normal. However, developers spend more time to code and to change on codes. Sometimes the developers may even need to extra jobs. They are disadvantages of Hadoop Map-Reduce. The disadvantages of Hadoop Map-Reduce are not in Apache Pig and Apache Hive. Development time is less in Apache Pig when compared Hadoop Map-Reduce.

In this section, rather than comparison, we wanted to show how to contribute new methods to Apache Pig and how to make a framework from a method.

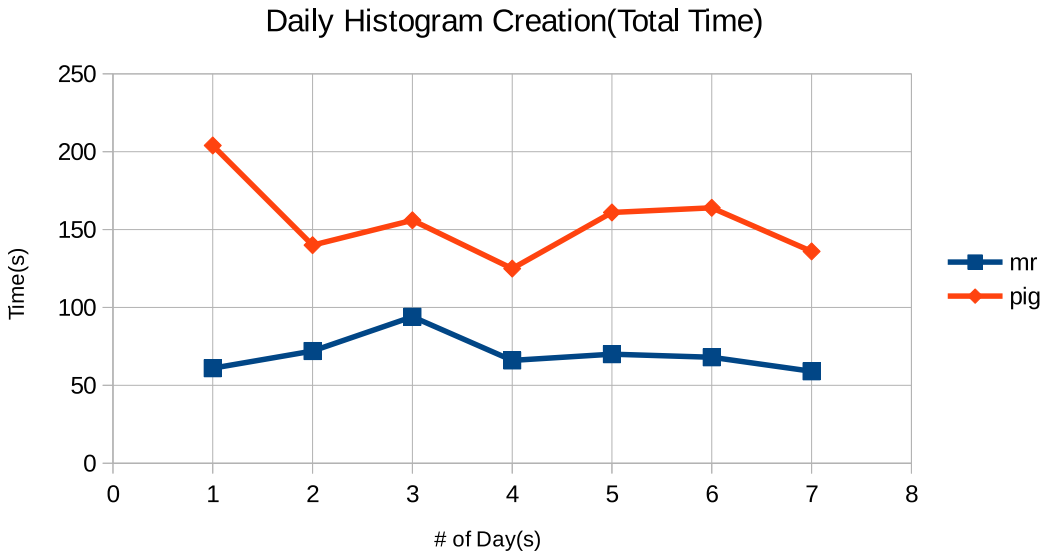



**Figure** 8.4 Running time against daily constructed histograms

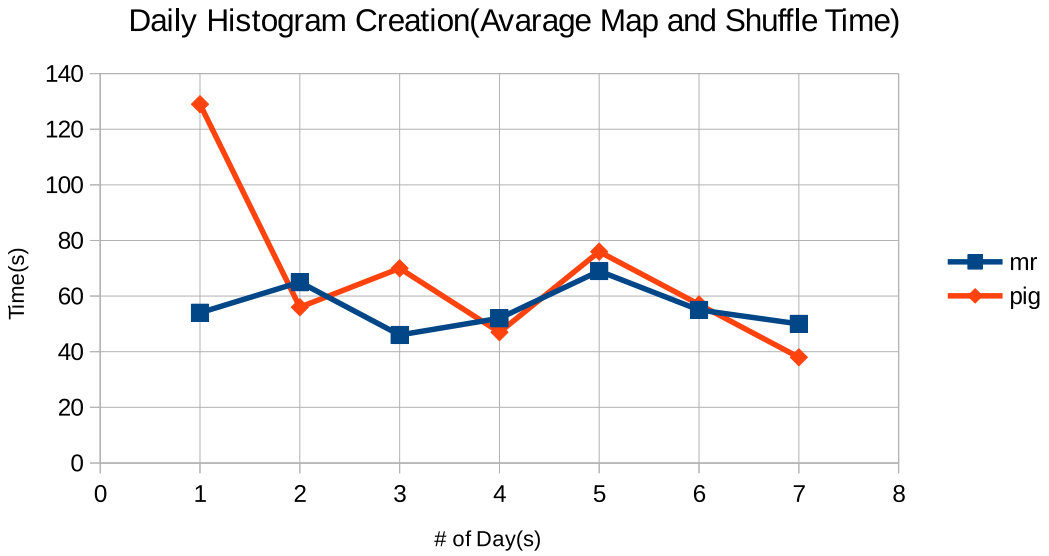



**Figure** 8.5 Avarage map and shuffle time against daily constructed histograms

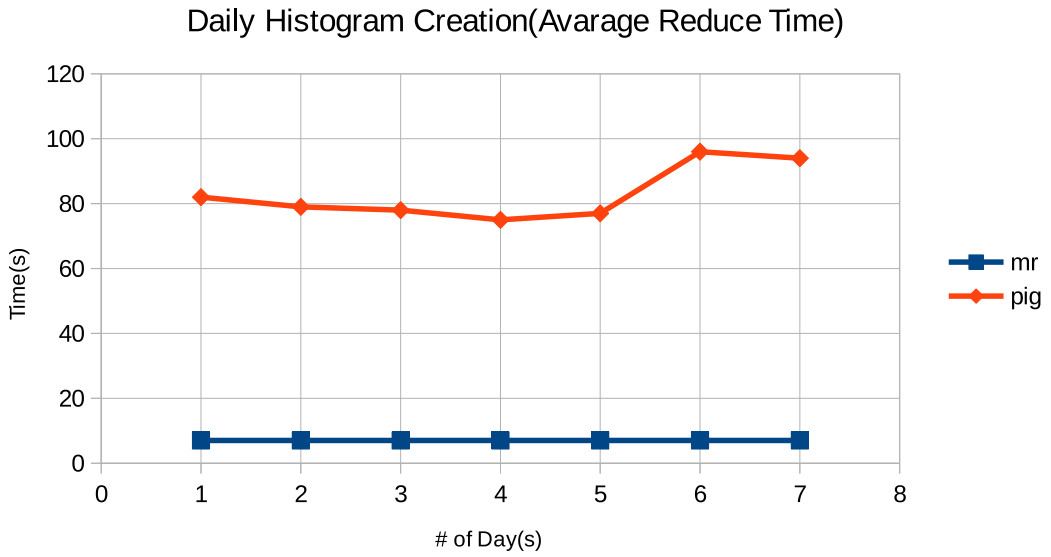



**Figure** 8.6 Avarage reduce time against daily constructed histograms

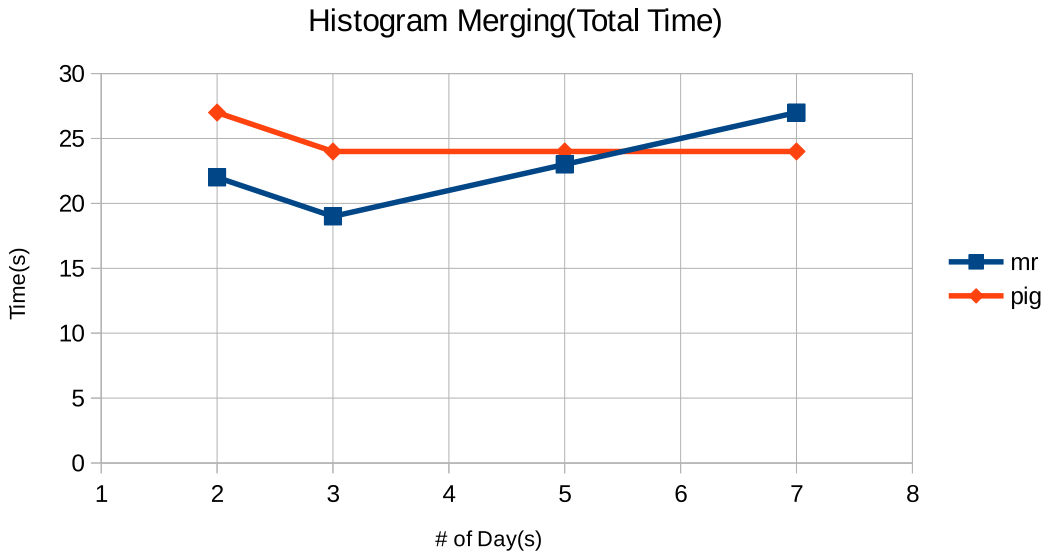



**Figure** 8.7 Running time against merged # of day(s)

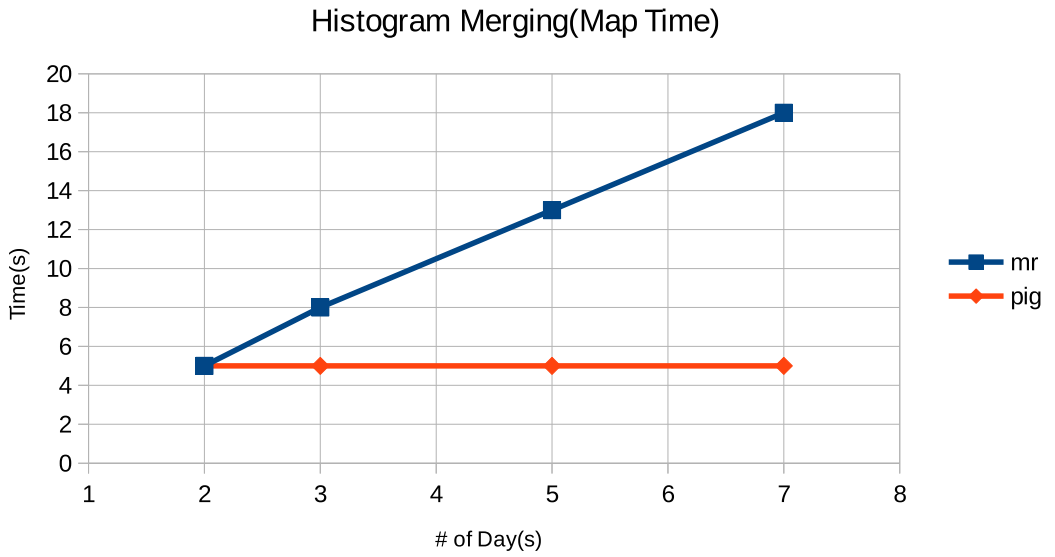



**Figure** 8.8 Avarage map time against merged # of day(s)

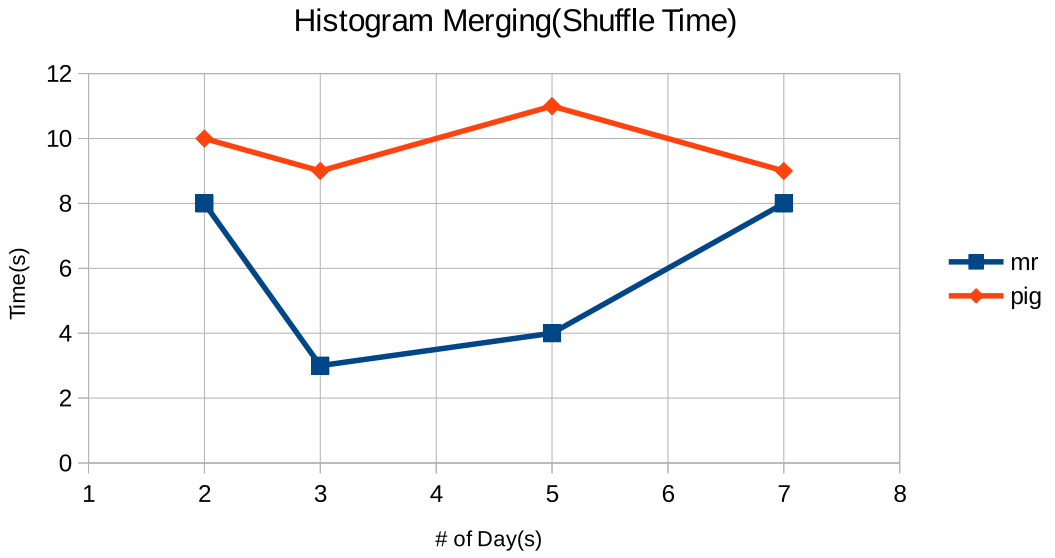



**Figure** 8.9 Avarage shuffle time against merged # of day(s)

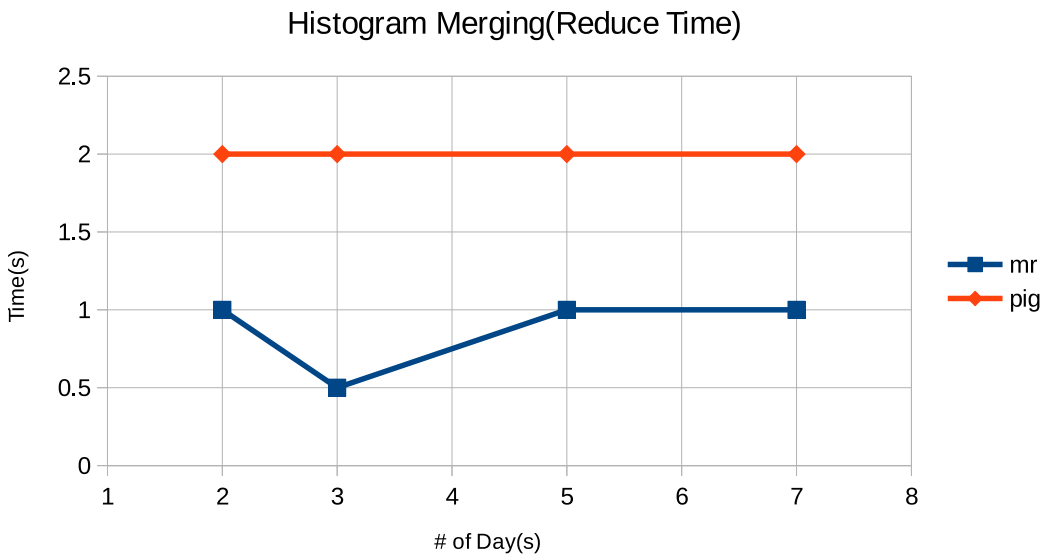



**Figure** 8.10 Avarage reduce time against merged # of day(s)

# CHAPTER 9

## CONCLUSION

In this thesis, we proposed a novel approximate equi-depth histogram construction method by merging precomputed exact equi-depth histograms of data partitions. The method is implemented on Hadoop to demonstrate how it is applied to real life problems. The theoretical calculations and the experimental results showed that both the bucket size errors and total size error of any bucket range are bounded by a predefined error set by a user in terms of $T$ and $\beta$. In particular, the experimental results run on both real and synthetic data show that the constructed histograms using the proposed method (*merge*) are more accurate than the tuple level random sampling (*tuple*) with a cost of offline run time. In addition to the proposed merged based histogram construction method, we also proposed a novel histogram processing framework for the daily stored log files. Besides, we contributed this methods to Apache Pig and created two demonstrations. This framework is crucial for fast histogram construction over a subset of a list of partitions on demand. The time complexity and the incrementally updated nature of the proposed method makes it practical to be applied over real life problems.

# REFERENCES


[1] Dionysios Logothetis, Christopher Olston, Benjamin Reed, Kevin C Webb, and Ken Yocum. Stateful bulk processing for incremental analytics. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 51–62. ACM, 2010.

[2] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1013–1020. ACM, 2010.

[3] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 996–1005. IEEE, 2010.

[4] Apache Software Foundation. Apache hadoop, 2008.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : a flexible data processing tool. *Communications of the ACM*, 53(1) :72–77, 2010.

[6] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, and Jörg Schad. Hadoop++ : making a yellow elephant run like a cheetah (without it even noticing). *Proceedings of the VLDB Endowment*, 3(1-2) :515–529, 2010.

[7] Alan F Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a high-level dataflow system on top of map-reduce : the pig experience. *Proceedings of the VLDB Endowment*, 2(2) :1414–1425, 2009.

[8] Alekh Jindal, Jorge-Arnulfo Quiané-Ruiz, and Jens Dittrich. Trojan data layouts : right shoes for a running elephant. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 21. ACM, 2011.

[9] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, volume 8, page 7, 2008.

[10] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad : distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[11] Avrilia Floratou, Jignesh M Patel, Eugene J Shekita, and Sandeep Tata. Column-oriented storage techniques for mapreduce. *Proceedings of the VLDB Endowment*, 4(7) :419–429, 2011.

[12] Yuting Lin, Divyakant Agrawal, Chun Chen, Beng Chin Ooi, and Sai Wu. Llama : leveraging columnar storage for scalable join processing in the mapreduce framework. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 961–972. ACM, 2011.

[13] Foto N Afrati and Jeffrey D Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 99–110. ACM, 2010.

[14] Spyros Blanas, Jignesh M Patel, Vuk Ercegovac, Jun Rao, Eugene J Shekita, and Yuanyuan Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 975–986. ACM, 2010.

[15] Alper Okcan and Mirek Riedewald. Processing theta-joins using mapreduce. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 949–960. ACM, 2011.

[16] Jorge-Arnulfo Quiane-Ruiz, Christoph Pinkel, Jörg Schad, and Jens Dittrich. Rafting mapreduce : Fast recovery on the raft. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 589–600. IEEE, 2011.

[17] Sai Wu, Feng Li, Sharad Mehrotra, and Beng Chin Ooi. Query optimization for massively parallel data processing. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 12. ACM, 2011.

[18] Shivnath Babu. Towards automatic optimization of mapreduce programs. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 137–142. ACM, 2010.

[19] Herodotos Herodotou and Shivnath Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *Proceedings of the VLDB Endowment*, 4(11) :1111–1122, 2011.

[20] Eaman Jahani, Michael J Cafarella, and Christopher Ré. Automatic optimization for mapreduce programs. *Proceedings of the VLDB Endowment*, 4(6) :385–396, 2011.

[21] Dawei Jiang, Beng Chin Ooi, Lei Shi, and Sai Wu. The performance of mapreduce : An in-depth study. *Proceedings of the VLDB Endowment*, 3(1-2) :472–483, 2010.

[22] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Stefan Richter, Stefan Schuh, Alekh Jindal, and Jörg Schad. Only aggressive elephants are fast elephants. *Proceedings of the VLDB Endowment*, 5(11) :1591–1602, 2012.

[23] Yahoo advertising.

[24] Yannis Ioannidis. The history of histograms (abridged). In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 19–30. VLDB Endowment, 2003.

[25] F. Emekci, S.P. Chakkappen, and U. Shaft. Determining a height-balanced histogram incrementally, March 5 2013. US Patent 8,392,406.

[26] Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. Improved histograms for selectivity estimation of range predicates. In *ACM SIGMOD Record*, volume 25, pages 294–305. ACM, 1996.

[27] Anna C Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, Sivaramakrishnan Muthukrishnan, and Martin J Strauss. Fast, small-space algorithms for

approximate histogram maintenance. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 389–398. ACM, 2002.

[28] Ying-Jie Shi, Xiao-Feng Meng, Fusheng Wang, and Yan-Tao Gan. Hedc++ : An extended histogram estimator for data in the cloud. *Journal of Computer Science and Technology*, 28(6) :973–988, 2013.

[29] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. In *VLDB*, volume 97, pages 466–475, 1997.

[30] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction : How much is enough ? In *ACM SIGMOD Record*, volume 27, pages 436–447. ACM, 1998.

[31] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Histogram construction using adaptive random sampling with cross-validation for database systems, 2001. US Patent 6,278,989.

[32] Surajit Chaudhuri, Gautam Das, and Utkarsh Srivastava. Effective use of block-level sampling in statistics estimation. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 287–298. ACM, 2004.

[33] Dhruba Borthakur. Hdfs architecture guide. *HADOOP APACHE PROJECT http ://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[34] Jeffrey Jestes, Ke Yi, and Feifei Li. Building wavelet histograms on large data in mapreduce. *Proceedings of the VLDB Endowment*, 5(2) :109–120, 2011.

[35] Phillip M Hallam-Baker and Brian Behlendorf. Extended log file format. *WWW Journal*, 3 :W3C, 1996.

[36] Burak Yıldız, Tolga Büyüktanır, and Fatih Emekci. Equi-depth histogram construction for big data with quality guarantees. *arXiv preprint arXiv :1606.05633*, 2016.

[37] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[38] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : simplified data processing on large clusters. *Communications of the ACM*, 51(1) :107–113, 2008.

[39] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin : a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[40] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive : a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2) :1626–1629, 2009.

# CURRICULUM VITAE

## TOLGA BÜYÜKTANIR

### PERSONEL INFORMATION

**E-mail :** tolgabuyuktanir@gmail.com

**Birthday :** 10.03.1992

**Birth Place :** Kağızman, TURKEY

**GENDER :** Male

### EDUCATION

**Graduate :** Yıldırım Beyazıt University, ANKARA, Computer and Electrical Engineering Department, Master Student(2014 September,Present)

**Interest :** Big Data Concepts, Hadoop Ecosystems, Internet of Things

**Master Thesis :** Equi-Depth Histogram Construction for Big Data with Quality Guarantees

**Undergraduate :** Erciyes University, KAYSERİ, Computer Engineering, 2014

### EMPLOYMENT

- GÖN DERİCİLİK, ANKARA, Freelancer Database Developer(2016 October, Present)
- Freelancer.com,Armut.com, Freelancer Java and Big Data Developer(2016 September,Present)
- Turgut Ozal University, ANKARA, Computer Engineering Department Research/Teaching Assistant (2014 October-2016 July)

### LANGUAGES

- **Turkish** - Native Language
- **English** – High level reading/writing/listening/speaking