Technische Universität Berlin

Big Data Management Project

Winter Semester 2018-2019

BDAPRO Project Report

Authors:
Syeda Noor Zehra Naqvi
Student ID: 406146
Sofia Yfantidou
Student ID: 405755
Todi Thanasi
Student ID: 405754

Supervisor:
Alireza Rezaei
Mahdiraji

November 5, 2019

Contents

1 Introduction

According to Forbes[1], in 2018, the adoption rate of IoT by all industries reached 45%, while Connected Health was amongst the top 10 IoT segments in 2018, holding 6% of the global share of IoT projects. Healthcare data is growing at rampant rates and is projected to exceed 2314 exabytes by 2020[2]. Iot is growing exponentially (See Figure 1) and resource consumption for storage and processing grows along with it. For example simply storing 500TB of data in an Amazon S3 bucket with infrequent data access would cost a company \$10000/month, or \$18500/month for standard S3 storage with frequent access[3]. These exorbitant costs create the need for solutions to achieve:

1. High data compression rates to decrease space requirements and processing time
2. Minimal quality loss to achieve accurate approximations

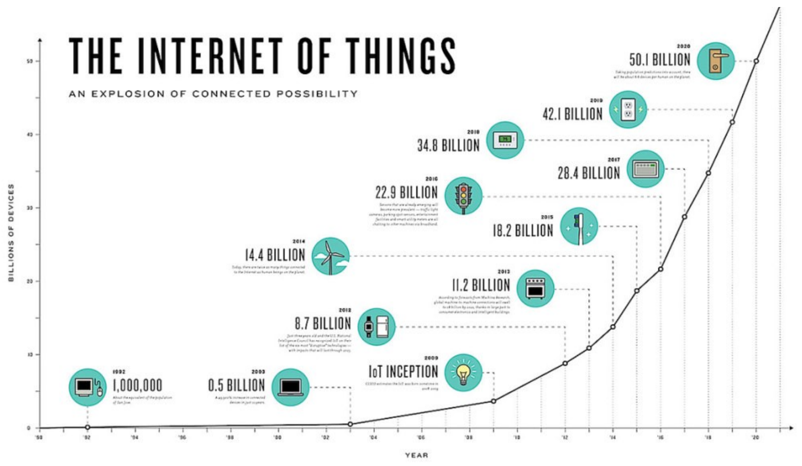


Figure 1: Expected growth of IoT [4].

If we think of an IoT data stream as a signal, what we need is a suitable representation of it, where we can discard its least significant parts and thus keep the original signal largely intact. This requires a transformation which separates the important parts of the signal from less important parts [6]. While Fourier Transform [1] might be the first thing to cross someone's mind, this transformation has only frequency resolution and no time resolution. This means that while we may be able to identify all the

[1] "10 Charts That Will Challenge Your Perspective Of IoT's Growth - Forbes." 6 Jun. 2018, `https://www.forbes.com/sites/louiscolumbus/2018/06/06/10-charts-that-will-challenge-your-perspective-of-iots-growth/`. Accessed 7 Mar. 2019.

[2] "Global Big Data in Healthcare Market to Reach \$68.75 Billion by 2025, Report BIS Research." 28 Mar. 2018, `https://www.prnewswire.com/news-releases/global-big-data-in-healthcare-market-to-reach-6875-billion-by-2025-reports-bis-research-678151823.html`. Accessed 7 Mar. 2019.

[3] "Amazon Simple Storage Service (S3) Pricing in the AWS GovCloud US Region" `https://aws.amazon.com/govcloud-us/pricing/s3/`. Accessed 7 Mar. 2019.

frequencies present in a signal, we cannot tell when they occur [5]. Moreover, Fourier Transform performs poorly in approximating sharp spikes in signals, which are pretty common in physiological data, such as Electrocardiograms (ECG), Arterial Blood Pressure (ABP), etc. [2].

An alternative to Fourier Transform for such signals is the wavelet technique, which has been considered very promising for data compression and query approximation [4]. The main difference between Fourier Transform and Wavelet Transform is that the latter is localized in both time and frequency whereas Fourier is only localized in frequency. Discrete wavelet transform, e.g., Haar Wavelet Transform (See Section 2.1) has already been successfully applied for the compression of ECGs [3], producing a synopsis of the original signal.

There are two categories for constructing wavelet synopses given an upper-threshold of the approximation error [4]:

1. **Bucket-Bound Synopses:** The goal is to construct a synopsis of at most B Haar wavelet coefficients to minimize the maximum total error.

2. **Error-Bound Synopses:** Given an error bound Δ, the goal is to find the wavelet synopsis with the smallest set of coefficients among all possible solutions, that would satisfy the error bound on the maximum error.

Error-bound synopses are preferred in scenarios that do not naturally adapt to Bucket-Bound synopses. For example, if we want to correlate patients' sampled physiological data from surgeries, a Bucket-Bound synopsis might result in unqualified samplings, due to the different duration of each operation. Thus, an Error-Bound synopsis might be preferable.

This project focuses on Error-Bound synopses. Specifically, we study the F-Shift algorithm (See Section 2.3 for more details), which is an improvement to the widely used Haar transform. Also, we provide implementation and comparison between its centralized and distributed version with different data and cluster characteristics, presented in Section 3 and Section 5, respectively. Finally, we discuss the challenges faced during the parallelization of the algorithm in Section 4 and provide some interesting conclusions and ideas for future work in Section 6.

2 Background

As mentioned in Section 1, the wavelet decomposition technique is present in many applications that require data compression or query approximation and is finding usage in several domains. The majority of existing wavelet algorithms depend on coefficients of synopses and their summaries. Also, their time and space complexity makes them ill-suited for data stream processing. The paper that we follow for our project [4] presents a new algorithm for constructing unrestricted, error-bound Haar wavelet synopses. The algorithm is named F-Shift and has linear time complexity and a $\log(N)$ space complexity. So, it is possible to apply such algorithm for stream data problems [4].

We give a general idea about Haar wavelet synopsis in Section 2.1, since it is the basis for the logic of the new F-Shift algorithm. In Section 2.2, we explain the basic concepts for a better understanding of the algorithm. While in Section 2.3 we explain the F-Shift algorithm in greater detail.

2.1 Haar Wavelet

The Haar wavelet is one of the most known and uncomplicated wavelet decomposition. The construction of such synopses consists of simply calculating averages and differences of the data points. So, for a given data vector D=[19, 17, 12, -4, 7, -1, -3, -7] the Haar wavelet for the first level contains two sets of values: a set of the averages of each two consecutive points [18, 4, 3, -5] and a set of the differences between them [1, 8, 4, 2]. The process continues recursively until a single point remains. We can easily reconstruct the initial data points by using the computed sets of values. The final set of averages and differences represents what are called Haar coefficients. The Haar wavelet decomposition values can represent an error tree where each of the nodes is a Haar coefficient, and the leaves are the original data points [4].

2.2 F-Shift Algorithm Concepts

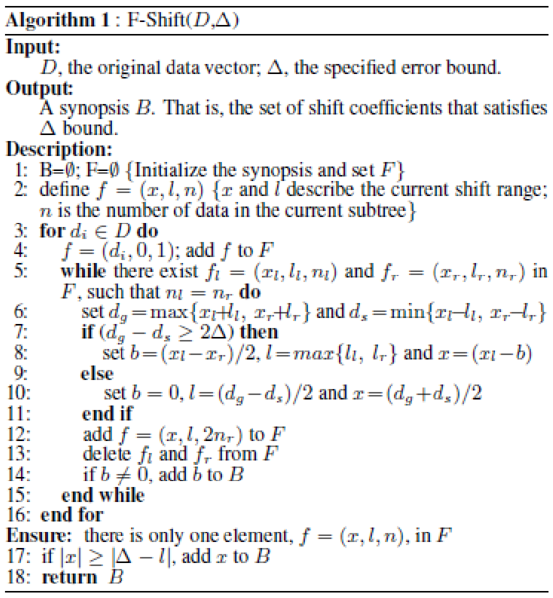
We explain below three main concepts that the F-Shift algorithm introduces: [4].
Shift Transformation: Let take one of the error Tree T coefficient c_i and a new s_i real number value. The s_i is called a shift transformation at the coefficient position c_i when it transforms the error tree T into T' and is still under the error bound Δ. The steps to build T' are:

1. Replace the c_i coefficient with $c_i - s_i$.
2. When the level $i > 0$
   - replace every leaf `d` in the left sub-tree $T_L(c_i)$ with $d - s_i$.
   - replace every leaf `d` in the right sub-tree $T_R(c_i)$ with $d + s_i$.
3. When the level $i = 0$, replace every leaf with $d - s_0$.

Data Scope: Starting from a sub-tree T' of the error tree T and a set of shift coefficients S, the data scope describes the data space of T' and it measures it as $max_{d'_i,d'_j \in T'<S>}|d'_i - d'_j|$ [4]. In simpler words, the data scope explains if a coefficient of a sub-tree needs to be shifted or not and which of the shifted coefficients s_i we should keep.
Shift Range: It specifies which of the c_i coefficients should be chosen and in what value range in order to be shifted as s_i coefficients. The formulas for the shift range are part of the F-Shift algorithm (lines 6 to 11) that Figure 2 shows.

2.3 F-Shift Algorithm

The terminology unrestricted error-bound synopses means that in a given error bound Δ, we construct a synopsis S_{op} with the smallest set of unrestricted coefficients that has an error smaller than Δ. The word unrestricted means that we are not limited in using only Haar coefficients, but they can have different values [4].



```
Algorithm 1 : F-Shift($D$,$\Delta$)
Input:
    $D$, the original data vector; $\Delta$, the specified error bound.
Output:
    A synopsis $B$. That is, the set of shift coefficients that satisfies
    $\Delta$ bound.
Description:
 1: B=$\emptyset$; F=$\emptyset$ {Initialize the synopsis and set $F$}
 2: define $f = (x, l, n)$ {$x$ and $l$ describe the current shift range;
    $n$ is the number of data in the current subtree}
 3: for $d_i \in D$ do
 4:     $f = (d_i, 0, 1)$; add $f$ to $F$
 5:     while there exist $f_l = (x_l, l_l, n_l)$ and $f_r = (x_r, l_r, n_r)$ in
        $F$, such that $n_l = n_r$ do
 6:         set $d_g = \max\{x_l + l_l,\ x_r + l_r\}$ and $d_s = \min\{x_l - l_l,\ x_r - l_r\}$
 7:         if ($d_g - d_s \geq 2\Delta$) then
 8:             set $b = (x_l - x_r)/2$, $l = max\{l_l,\ l_r\}$ and $x = (x_l - b)$
 9:         else
10:             set $b = 0$, $l = (d_g - d_s)/2$ and $x = (d_g + d_s)/2$
11:         end if
12:         add $f = (x, l, 2n_r)$ to $F$
13:         delete $f_l$ and $f_r$ from $F$
14:         if $b \neq 0$, add $b$ to $B$
15:     end while
16: end for
Ensure: there is only one element, $f = (x, l, n)$, in $F$
17: if $|x| \geq |\Delta - l|$, add $x$ to $B$
18: return $B$
```

Figure 2: F-Shift Algorithm [4].

The F-Shift algorithm has a bottom-up approach, and it constructs a synopsis starting from the data point d_0 until d_{N-1} as line 3 in Figure 2 shows. The algorithm starts by defining some data structures to keep intermediate values and perform calculations such as $f = (x, l, n)$ where x and l describe the current shift range and n is the number of data in the sub-tree. The core logic of the algorithm is in the lines 7-11 where it enforces the data scope by using the left ($f_L = (x_L, l_L, n)$) and right ($f_R = (x_R, l_R, n)$) sub-trees. After calculating the data scope values, we check if the sub-tree is s-bounded or not (line 7). In case, the sub-tree is s-unbounded we need to compute the new shift transformation because the algorithm should guarantee that the final synopsis is error bounded. The algorithm recursively processes all the initial points and intermediate results until it remains the last root coefficient. Line 17 of the algorithms check that the last c_0 coefficient should be shifted or not in a new s_0 value. The algorithm and its properties guarantee that the final set of shifts coefficients is unrestricted to Haar coefficients and Δ error bounded.

3 Methodology

This section describes the logic of F-Shift algorithm's parallelization, including a Flink workflow visualization, as well as the reconstruction logic, which was implemented for benchmarking purposes.

3.1 Distributed F-Shift

We implemented F-Shift in Flink as follows (See Figure 4 for ID reference):

1. We have a text file with time series data as a data source. Each line consists of a (timestamp, measurement) pair. $\rightarrow$ ID=1 and ID=2 in the workflow

2. (Optional) We add a throttle after each fixed-size batch of points (batch size is user defined) per worker, as a workaround to Flink's iteration bug[5]. This functionality can be found in the AddDelay class and its stateful map function. $\rightarrow$ ID=3 in the workflow

3. We convert each file line into a custom Point Java object using the mapToRealValue class. $\rightarrow$ ID=4 in the workflow

4. We then use a Flink iteration to implement the algorithm's main logic. In each iteration, we perform the following: $\rightarrow$ ID=7 in the workflow

   (a) Define a composite key for each point in the stream. The composite key includes the level n and the result of the calculation $timestamp - (timestamp\%(n \times 2))$. For every subsequent level, the timestamp of the newly produced point is the minimum of its two parents. Figure 3 provides a visualization of how the composite keys are computed. Composite keys are critical in our application because they enable in-order processing of an unordered stream. As can be seen in the figure above, even though the elements are out-of-order, they are processed in an orderly fashion, which solves the problems of overlapping ranges and disproportionate work allocation, which we describe in Section 4.

   (b) We use a count window to group points in pairs. Every two points in a pair now have consecutive timestamps due to the use of the composite key. This grouping is necessary, as F-Shift works with pairs of points (f_L and f_R as explained in Section 2).

   (c) We use the reduce function of reduceToOnePoint class to combine the two points (f_L and f_R) into one (f). The logic implemented in the reduce function is the same as the algorithm in Section 2.3 (lines 6 to 11). Note here that inside each point, we also store a shift coefficient (b) produced by the two points. b can either be non-zero or be zero depending on whether the condition $d_g - d_s \geq 2\Delta$ is satisfied or not, respectively (line 7).

5. The points produced into each iteration are then passed to the next iteration round (for the next level), waiting to be combined. At the same time, they are sent outside of the iteration as well, where the shift values are being extracted utilizing the flatMap function of the ShiftValues class. $\rightarrow$ ID=-1 in the workflow

[5] "[FLINK-2470] Stream Iteration can Hang after some data - ASF JIRA - Apache issues." 22 Oct. 2015, `https://issues.apache.org/jira/browse/FLINK-2470`. Accessed 7 Mar. 2019.

6. Finally, we write the shift values, along with their ranges and corresponding level to the output file. → ID=-1 in the workflow

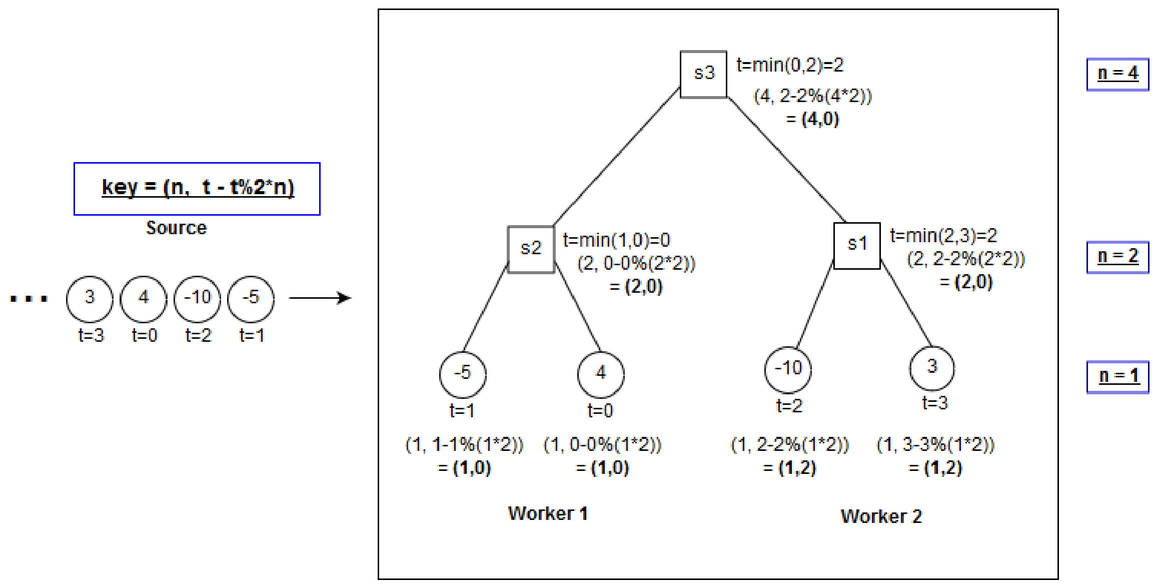


Figure 3: Visualization of how composite keys are produced.

The Flink workflow of the implementation above can be seen in Figure 4.

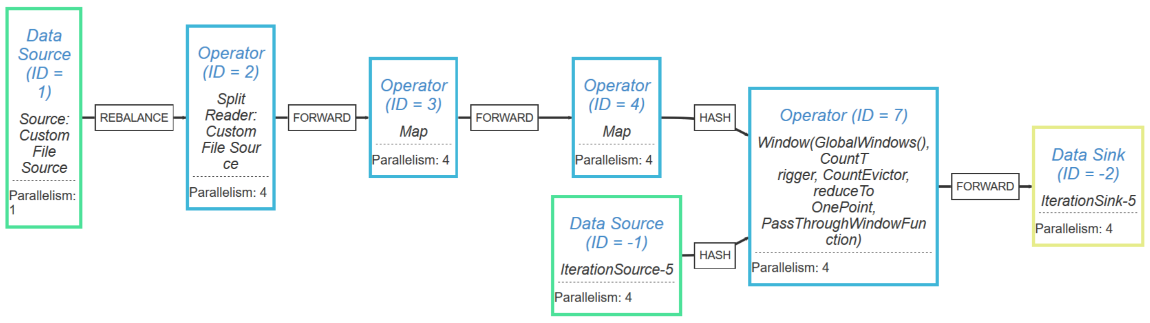


Figure 4: Flink's distributed workflow chart.

3.2 Distributed Reconstruction

By executing the F-Shift algorithm implementation, as described in the previous section, we obtain an output file containing the shift coefficients. A file line contains a tuple (coefficient, level, rangeFrom, rangeTo). Of course, such file is useless if we are not able to reconstruct the approximation of the original data using these coefficients. That's why we also implemented the reconstruction logic, to access the quality of our implementation. The reconstruction implementation works as follows:

1. We parse the file containing the shift coefficients, and we identify the minimum and maximum timestamp, i.e., the reconstruction range, as well as construct a

map structure which for each level contains only the shift coefficients of that level. This functionality is implemented in reconstructFromFile function.

2. Given the map structure of the previous step, we reconstruct each point in the reconstructFromMap function as follows:
   (a) For every level, we check whether the timestamp at hand is included in the range of any of the level's coefficients.
   (b) If yes, we check whether the timestamp is below or above the middle of the range. If it is below, we add the shift coefficient to the reconstructed value. If it is below, we subtract it. The root level is an exception where we always add the coefficient.
3. Having obtained the reconstructed value we can now write it in the output sink.

4 Challenges

Following are the main challenges we faced while working on this project.

- The initial challenge was to understand the base research paper and F-Shift algorithm [4], as there were many aspects which were missing in this paper. The approach in the paper even for the centralized case, does not mention how the data is going to be reconstructed from the shift values and what additional information is required to be saved to reconstruct the original data. The basic concepts are also not explained theoretically, and it only provides the mathematical formulas for them without stating what they are based on.
- Another major challenge was to figure out a way to distribute the tasks in a well-balanced way across different machines and reducers as the paper describes only the centralized approach and does not talk about how it can be executed in a distributed environment. Initially, we were using the size of subtree n as described in section 2 (reflecting the level in the tree) as key for the map-reduce problem. This was creating a bottleneck by sending all data points to one reducer in the first step even though it was well distributed in the later steps. We solved this problem by coming up with a composite key as explained in section 3.
- This algorithm requires to maintain the order of incoming data points to construct the tree and create shift values so that the data can be reconstructed. This was also a challenge, as it is hard to enforce order in distributed programming. Our choice of the key also catered this problem.
- We faced multiple problems because Flink iteration is needed in our program. Figure 2 shows that the last step of the algorithm is outside the iteration, so the algorithm needs to be aware of the last execution step. We solved this problem by using a timeout for iteration when nothing further is being received

in the stream for a certain amount of time and used a map function outside the iteration which checks if the data coming outside of iteration indeed is from the root of the tree to execute the required step as explained in 3.

- Another problem with using Flink iterations is the Flink iteration deadlock bug[6]. We tried solving the issue by cherry picking a hack solution proposed by Gevay, Gabor Etele from DIMA[7]. We also tried a solution implemented by one of the teams working on this problem as BDAPRO project. Additionally, we tried adding delays between data ingestion. However, we were unable to solve this issue completely.
- Developing a method for reconstruction of the approximated original data from the shift values was a challenge because as mentioned earlier the paper itself does not talk about it at all.
- As this was our first big project with Flink, it was also a challenge for us to understand the internals of Flink.

5 Experiments & Discussion

This section provides a detailed description of the benchmarking of the centralized and distributed F-Shift algorithm and a discussion of the results. We evaluated different aspects of the implementation in terms of result quality (compression rate, error boundness, synopsis quality) and time efficiency (speedup for different data sized and parallelism).

5.1 Benchmarking Setup

For running the experiments, we used a cluster of 8 IBM machines with 62 GB memory and 42 cores each. The operating system is Fedora 27 with Linux version 4.18.12-100.fc27.ppc64. The available disk space is 1TB. The software used to run the application is Flink-1.7.1.

The experiments are conducted on synthetic datasets that follow Zipfian and uniform distribution. We generate several datasets with a different number of points and document it properly in each of the experiments. The Zipfian datasets contain values generated with a Zipfian parameter of 2.0, which means that the dataset has a highly skewed distribution. The uniform datasets contain values following uniform distribution in the range [0,1000).

[6] "Flink Iteration Bug with Big Data." 3 Aug. 2015, `https://issues.apache.org/jira/browse/FLINK-2470`. Accessed 7 Mar. 2019.

[7] "Hack the two Flink iteration deadlocks." 25 Jun. 2017, `https://github.com/ggevay/flink/commit/c0957019059cb3f6098d01ed74091ccf17257576`. Accessed 7 Mar. 2019.

5.2 Compression Rate

The evaluation of the compression rate in the case of our algorithm is critical since one of the features that such algorithms should offer is a good compression of the data. We use the two synthetic datasets with Uniform and Zipfian distribution with a total number of initial data points of $N = 2^{24}$ (approximately 16.7M points).

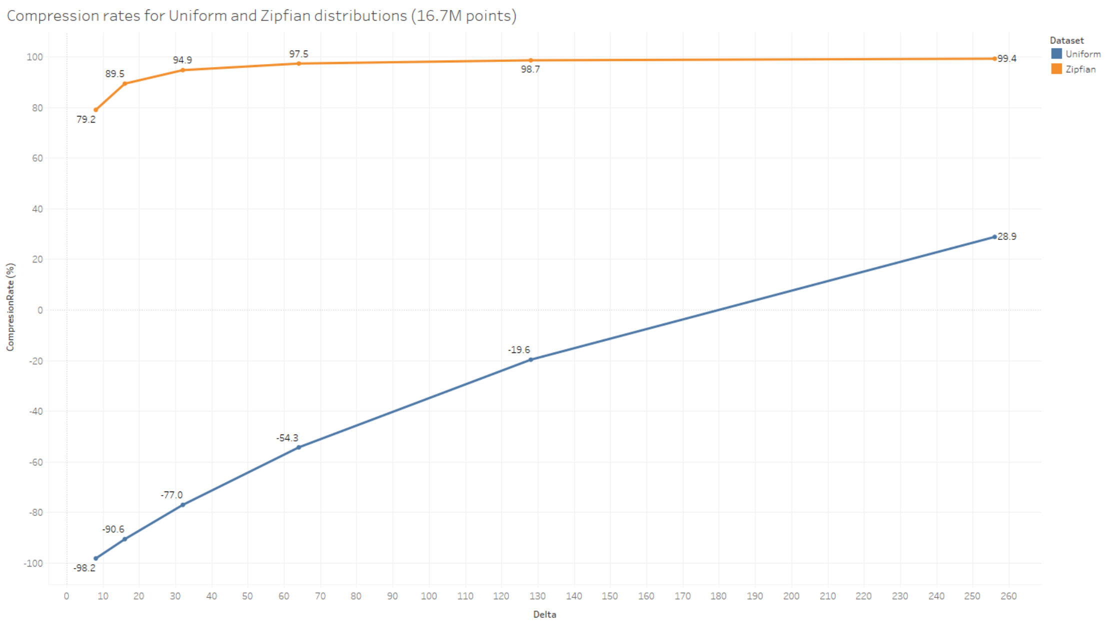


Figure 5: Compression rates for Uniform and Zipfian distributions $N = 2^{24}$ (16.7M points).

Figure 5 shows the result of such experiment. We can observe that there is an enormous compression rate for the dataset that follows Zipfian distribution. It has a compression rate of 79.2% for a minimal $\Delta = 8$ and it achieves more than 90% for a $\Delta \approx 20$. The case of the uniform dataset is not offering the same compression rates. We observe that for small Δ the compression rate is negative. The negative value means that the shifts file size is bigger than the dataset itself. The bigger file size does not come due to more shifts values than actual points because this is not possible but from the fact that we store metadata information in addition to shifts values. The metadata is needed to reconstruct the initial data points because using only the shifts values is not possible (for reconstruction of original data). The uniform dataset starts to have a positive compression rate for Δ bigger than 180. However, the compression rate is not very high as in the case of Zipfian distribution.

The compression rate experiment shows that significant differences between two consecutive points impact the efficiency of the algorithm in terms of compression rate. The Zipfian distribution contains a lot of consecutive 1 values that make it suitable for the F-Shift algorithm. However, not only Zipfian distribution should fit for our algorithm but whatever similar dataset that contains consecutive points which do not have significant value fluctuations.

5.3 Error-Boundness

While the centralized approach is designed to be error-bounded, as it strictly follows the F-Shift algorithm, we had to verify that the same applies to the distributed approach. We inspected the distributed execution to verify that the resulting approximations were error-bounded. We used both Zipfian and uniform distribution as input data for this experiment with two synthetic datasets of 2^{18} points each. For the distributed execution we used parallelism of 20.

Figures 6 and 7 show the results of the experiment. The histograms represent the distribution of the reconstruction error (actual value-reconstructed value). To calculate this reconstruction error we had to implement a separate component that is responsible for the reconstruction of the time series data given a set of shift values produced by the F-Shift algorithm as explained in Sections 2 and 3. We can see that in both cases the error is less or equal to the error bound of $\Delta = 8$. This means that parallelizing the F-Shift algorithm did not affect its property of producing error-bound synopses.

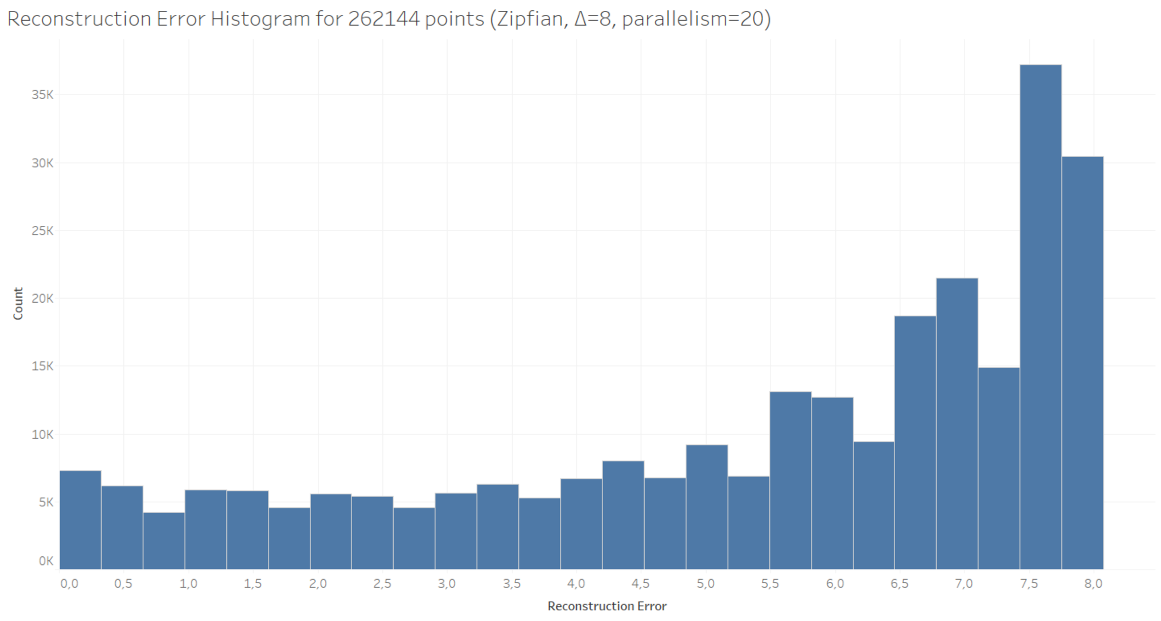


Figure 6: Reconstruction Error Histogram for 262144 points (Zipfian, Δ=8, parallelism=20).

Note that the difference in the distribution of reconstruction error between Zipfian and uniform input data set can be attributed to the significantly higher compression rates of the Zipfian distribution for the input data size of 2^{18}. Figure 8 shows the actual, uniformly distributed input data in orange and the reconstructed data, based on the produced shift coefficients, in blue. We can see that the reconstruction is of very high quality.

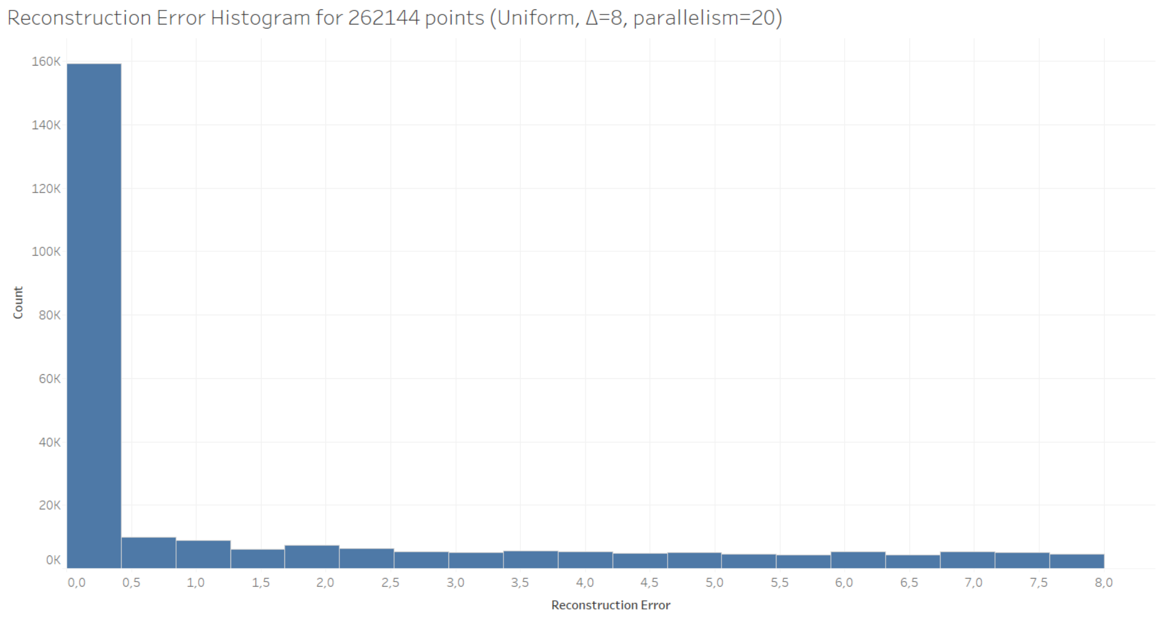


Figure 7: Reconstruction Error Histogram for 262144 points (Uniform, Δ=8, parallelism=20).

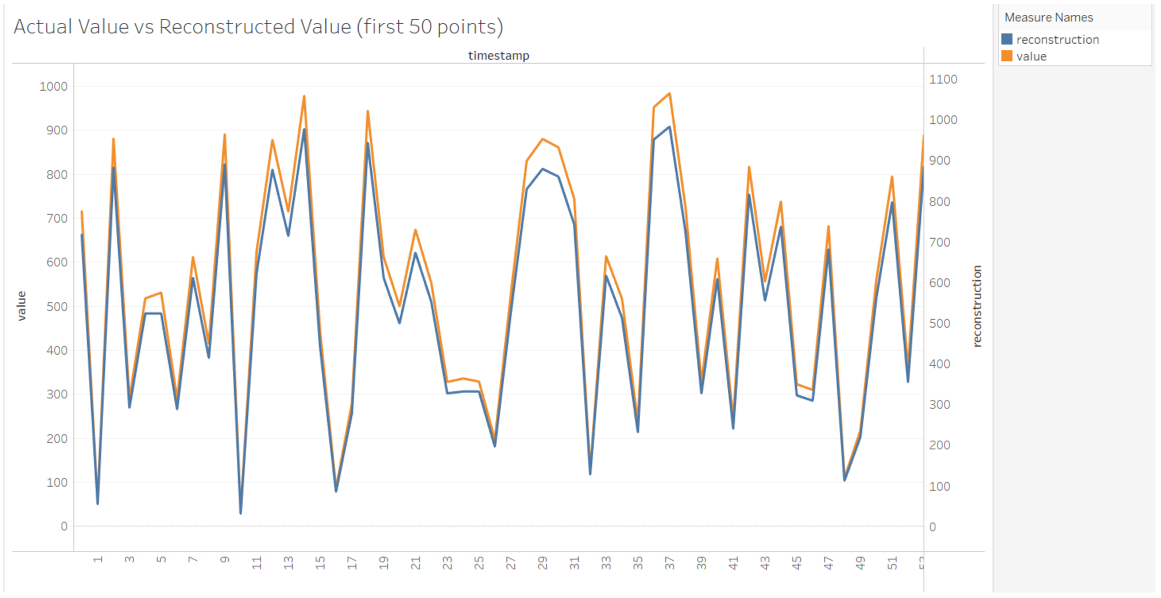


Figure 8: Actual vs Reconstructed Data for the first 50 points (Uniform, Δ=8, parallelism=20).

5.4 Centralized vs Distributed: Performance

We compared the centralized execution with distributed execution to calculate the gain in performance introduced by the distributed approach. We used Zipfian distribution for this experiment as the real world data is usually reflected better by Zipfian distribution. We experimented with different data sizes up to $N = 2^{24}$ data points (approximately 16.7M points).

Figure 9 shows the results of this experiment. The number of points are plotted on the x-axis and time in seconds is plotted on the y-axis. The orange line represents distributed execution with parallelism of 20, and the blue line represents the centralized execution using only 1 parallelism. The difference in performance in terms of time between centralized and distributed approach is quite clear. The distributed execution is up to 95% faster than the centralized execution. After execution with approximately 4M points, we stopped execution for centralized version as it was taking too long to finish.

The performance gain by the distributed implementation is quite noticeable and execution time is very critical in case of streaming as the data arrival rate is much faster.

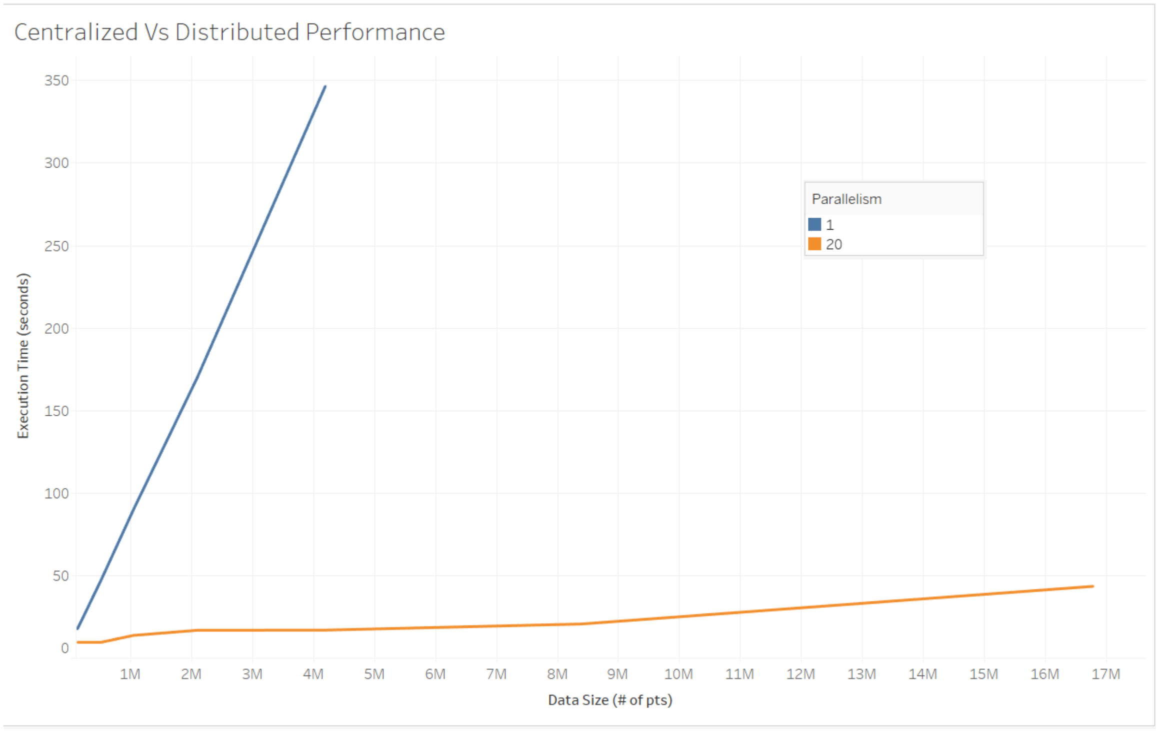
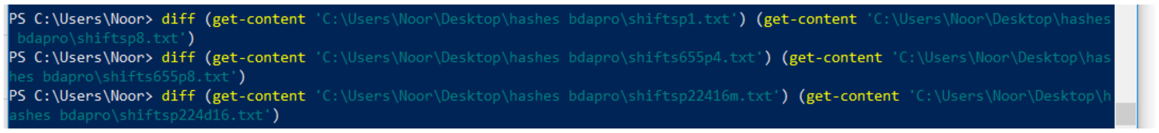


Figure 9: Execution time of centralized vs distributed approach for different data sizes. The blue line is shorter, because the centralized execution was either failing or taking exceedingly long.

5.5 Centralized vs Distributed: Generated Shift Values

We compared the results (shift values) produced by centralized (parallelism 1) and distributed (parallelism > 1) execution for the same input data set. We used the diff command on output files as shown in figure 10 to compare the results. We observed the difference of files generated by the same input data points, but different parallelism was null. This implies that shift values produced by centralized and distributed execution are the same. This proves that we do not lose any quality at all as we increase the parallelism. We use diff command instead of hash values of files for comparison as the order of the content inside the files can be different which could result in different hash values. However, the order of shift values in the file is not essential as it is not required for data reconstruction. So using parallel execution, we can improve performance keeping the quality (shift values) precisely the same.



```
PS C:\Users\Noor> diff (get-content 'C:\Users\Noor\Desktop\hashes bdapro\shiftsp1.txt') (get-content 'C:\Users\Noor\Desktop\hashes
 bdapro\shiftsp8.txt')
PS C:\Users\Noor> diff (get-content 'C:\Users\Noor\Desktop\hashes bdapro\shifts655p4.txt') (get-content 'C:\Users\Noor\Desktop\has
hes bdapro\shifts655p8.txt')
PS C:\Users\Noor> diff (get-content 'C:\Users\Noor\Desktop\hashes bdapro\shiftsp22416m.txt') (get-content 'C:\Users\Noor\Desktop\h
ashes bdapro\shiftsp224d16.txt')
```

Figure 10: Command to check difference between shift values of centralized and distributed execution.

6 Conclusions and Future Work

Data is growing bigger, and there is a need for scalable approaches for data processing to keep up with the pace. F-Shift algorithm proposes a way to achieve error bound data compression and summarization. This tackles part of the problem in terms of compressing large data sizes into smaller representations (shift values). However, it is also needed to parallelize this solution to achieve faster results. We implemented a distributed solution of the F-Shift algorithm using Flink. We achieved well-balanced distribution across all workers.

We also proposed and implemented a way of reconstruction of original data by these shift values keeping the data error bound. With our distributed approach we gained up to 95% speedup, keeping the quality of data intact. We also achieved a compression rate of 70 to 90% with a minimal error value.

However, the current solution only supports time-series data streams. We looked into turnstile streams as well, but we faced specific problems like the emergence of new shift values with updates. Another major issue while adopting this approach for turnstile streams is that we only keep the shift values, and when the updates in the original dataset come, these updates need adjustment in shift values without having to reconstruct the original data. These problems were out of the scope of our project as well as the time constraints that we had. In terms of future work, further investigation can be done in this direction to adopt our solution for turnstile streams.

References

[1] Bracewell, R. N., and Bracewell, R. N. *The Fourier transform and its applications*, vol. 31999. McGraw-Hill New York, 1986.

[2] Graps, A. An introduction to wavelets. *IEEE computational science and engineering 2*, 2 (1995), 50–61.

[3] Ko, S.-i., Choi, J.-s., and Kim, B.-h. Ag ramakrishnan and s. saha,“ecg coding by wavelet-based linear prediction,”. *IEEE Trans. Biomed. Eng 44* (1997), 394–402.

[4] Pang, C., Zhang, Q., Hansen, D., and Maeder, A. Unrestricted wavelet synopses under maximum error bound. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (2009), ACM, pp. 732–743.

[5] Valens, C. A really friendly guide to wavelets. *ed. Clemens Valens* (1999).

[6] Zorin, D. Lecture notes: Introduction to Wavelets, September 1997.