Learning based approaches to achieve better statistics in Big Data

by

Parag Paul

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 6, 2023

Keywords: statistics, accuracy, BigData, Edge Computing, databases, query optimizer

Approved by

Wei-Shinn Ku, Chair, Professor of Computer Science and Software Engineering
Haiquan Chen, Associate Professor of Computer Science and Software Engineering
Saad Biaz, Professor of Computer Science and Software Engineering
Tung Nguyen, Assistant Professor of Computer Science and Software Engineering
Yang Zhou, Assistant Professor of Computer Science and Software Engineering

Abstract

Statistics in Big data has multiple applications. There are industrial systems that depend on quick and accurate insights into possible anomalies and irregularities on a large scale, for them to take better business decisions.

Query Optimizers in Relational Database Management Systems(RDBMS) have traditionally used some form of sample based sketches or histograms to do cardinality estimations for join outputs and other predicates involved in the nodes of a SQL query. These approximations allow the optimizer component to find out near optimal join order for a tree of multiway joins. They also help in designing and planning in advance for the system resources like memory, number of cores(more relevant in modern cloud native configurations that utilize server chips with more than 100 cores).

Other than query optimizers, optimal statistics can also be used to answer questions within some error guarantees. Many *Big Data* and *Decision Support Systems(DSS)* need the ability to give faster turnaround time. With the advent of *Industrial Internet of Things(IOT)*, Edge devices and Autonomous systems, it is important that we can build better statistical models that work within the constraints of space available, memory requirements, error tolerance and speed of computation. The explosion of data has reached a point where it has already surpassed the growth rate of compute and memory capabilities which have not grown as fast.

We hereby propose multiple solutions and approaches to solve problems that are inflicting big data systems. We solve two of the problems related to statistics generation in such production systems. First we achieve a consistent level of estimation errors for a given workload, by creating statistics tied to the reduction of overall error in the system. Statistics by definition is an approximation of the actual information and hence is prone to errors when queried. The second problem we solve is that of being able to generate statistics in distributed streaming systems with limited resources in a streaming fashion. Once we are loaded with the power of fast and accurate statistics, we show the impact of optimized workloads on the power consumption metrics of such a system.

We solve the accuracy problem using regression algorithms to build statistical histograms that moves the state-of-the-art pivot of *q-error* or maximal estimation error to that of overall error by observing the slope and intercept of the errors. By minimizing the regression line of estimation errors, we guarantee a overall smoother workload than previous approaches that pivoted their solution towards reducing the maximal error. We present scenarios where this could be ineffective and result in larger error margins on the average workload, but provide lower maximal error guarantees.

We introduce two new metrics, **Q-Regression** pair**(Slope and Intercept)** of the regression line on the estimation errors and **QRegrArea**(area under the regression line). The lower the **QRegrArea**, the lower the overall estimation errors.

We prove experimentally that our algorithms are indeed effective on real workloads and various mathematical distribution families *(Normal, Uniform, Pareto, Random, Laplace, Cauchy, Zipfian)*. The dataset used from real world encompass census data and public information from Big Data sets.

For the second problem, we propose another algorithm, named as **AUPASS**. The algorithm can be used to create statistics using low footprint in terms of resources, in a streaming fashion, with the ability to losslessly merge the intermediate statistics in a distributed system, so that the summary view on the control node can be generated in a meaningful fashion. This has immense impact for modern day *Industrial Internet of Things* where the compute is limited and memory requirements are stringent. The algorithm uses sketch based summaries that can be integrated across nodes without loss of critical information. Algorithms like *Count Min Sketch*, *Hyperloglog*, *Reservoir Sampling* and *KLL* Sketches have been utilized to generate the summaries.

The results show that the proposed algorithms make a significant improvement over the state of the art. The combined power of the two independent proposals have the singular affect of being memory/compute efficient, stateless and energy efficient. The results of the experiments demonstrate that a better equipped optimizer will create better execution plans and eventually lower the energy requirements of any given system.

Acknowledgments

First of all, I will like to thank **Dr. Wei-Shinn Ku** for the guidance and the motivation, the regular checking in that he provided throughout the tenure. It has been a long journey for me. Right after my daughter went through the two long years of cancer treatment, I took a call. I was working for **Microsoft** at that point of time. I was having a lot of questions on the direction to be taken in life, now that I had seen death closely for so long. A doctorate was my dream since childhood and I always wanted to pursue that.

I realized that there will never be a good time.

As written above, the support system I have with my wife **Sancheeta Paul** and my daughter **Parissa Paul**, who I hope will someday come to terms with the reality of her cancer journey and how it inspired her father to go all the way and complete the Doctorate while working at Microsoft, that support system has kept me alive during an arduous battle where I even had ideas of giving up, but the original promise is what made me continue in the process. My family is the sole reason that I have come to this juncture in my journey. I can never put this to words and this is not necessarily the only way to show my gratitude, I promise. I remember my father **Pijush Kanti Paul**, a man of extreme memory and intellect, who acknowledged that troubling times made us let go of opportunities of higher education in the past, but the week he left us, he had an epiphany of sorts, that I will be able to complete my education in the future. My mother **Mithu Paul** is the last of my circle of trust and she has been a rock that I never can do without. Tireless hours of work

and dedication to family, that is what defines her and that is what defined my work ethics and professionalism.

I realized, I will always be gasping for time. I will always be working for some or the other company as technology will keep motivating me to solve more real world problems and that might stop me from ever taking the right steps needed to create the orientation that is needed to do proper research.

Despite all the conundrum, I decided that it was time for my PhD. It was during the **SIGSPATIAL** conference in **Seattle** back in 2018, that I met with **Dr. Ku**. I was committed to the idea since day one and now that I am typing my dissertation, I do want to bring to light that some decisions are made with impulse, some go with the flow but for me the decision was made. I knew this was the only direction that I have been thinking about for the last 5 years of my life. For me it was a life changing decision. To take myself out of the comfort zone of working in a company like **Microsoft** that had really good work life balance and I had multiple projects lined up and waiting for me.

I had worked less during the tumultuous medical journey in the past year, and so I was expected to work on them as much as possible. Despite those challenges, and my daughter needing close monitoring through her fifth birthday, I chugged along and **Dr. Ku**'s contributions in this journey cannot be put into simple words.

Next I would like to thank **Dr. Xiao Qin** for his invaluable always-on guidance in the process, starting right at the point where I called the students office for the first time back in the end of 2017, to guiding me through the process of actually applying and recommending **Dr. Ku** as a potential guide. His guidance and constant inputs were crucial.

I will like to extend my thanks to **Dr. Haiquan Chen** or **Prof. Victor** as a guiding light in the whole academic process. His kind words at opportune times set the ball rolling at various points in time. Through the conference rejections, his words were simple and gave me the light at the end of the tunnel.

Next and one of the most important mentors for me is **Asst. Prof. Wenlu Wang**, who helped a lot with my research.

Next I will like to thank **Dr. Saad Biaz** for the valuable inputs that he gave to me during the proposal defense. It was important for me to understand the process and his timely inputs were crucial.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A decade ago, the global data started to grow exponentially. There are no signs of slowing down. The situation has been been aggravated with increase in adoption of digital devices, government initiatives, Internet of Things and the hunger for data for machine learning in the post pandemic arena. The impact of this is yet to be fully quantified.

The pandemic saw the adoption of digital tools en masse. Corporations had to adapt fast for the changing requirements as the workforce imbibed the art of working from home. A decade ago, the network infrastructure and the data handling capabilities of that era,

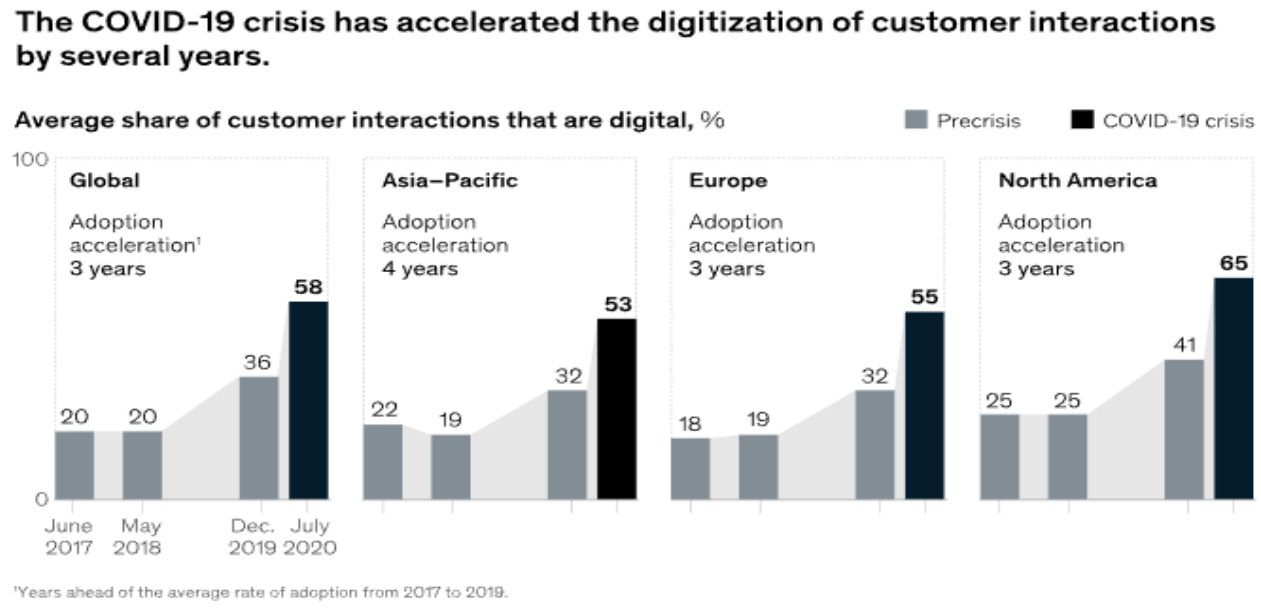


Figure 1.1: The pandemic has accelerated the digitization of the customer interactions by half a decade on average.

could not have met with the challenge as posed by the pandemic. The same pandemic, had it played out in 2010, could have been put the world productivity at the mercy of crippling infrastructure and data processing capabilities.

The limits of the technological infrastructure being pushed [12] was followed by expansion of the data requirements almost immediately. This meant that queries had to run faster in order for the throughput requirements to be met. For the already hyperoptimized systems, new fundamental changes in the processes had to follow. Things started to grow at government scale, when they realized that they could not stay far behind in the digitization processes. What was stalled for years, were suddenly allowed through the gates. This

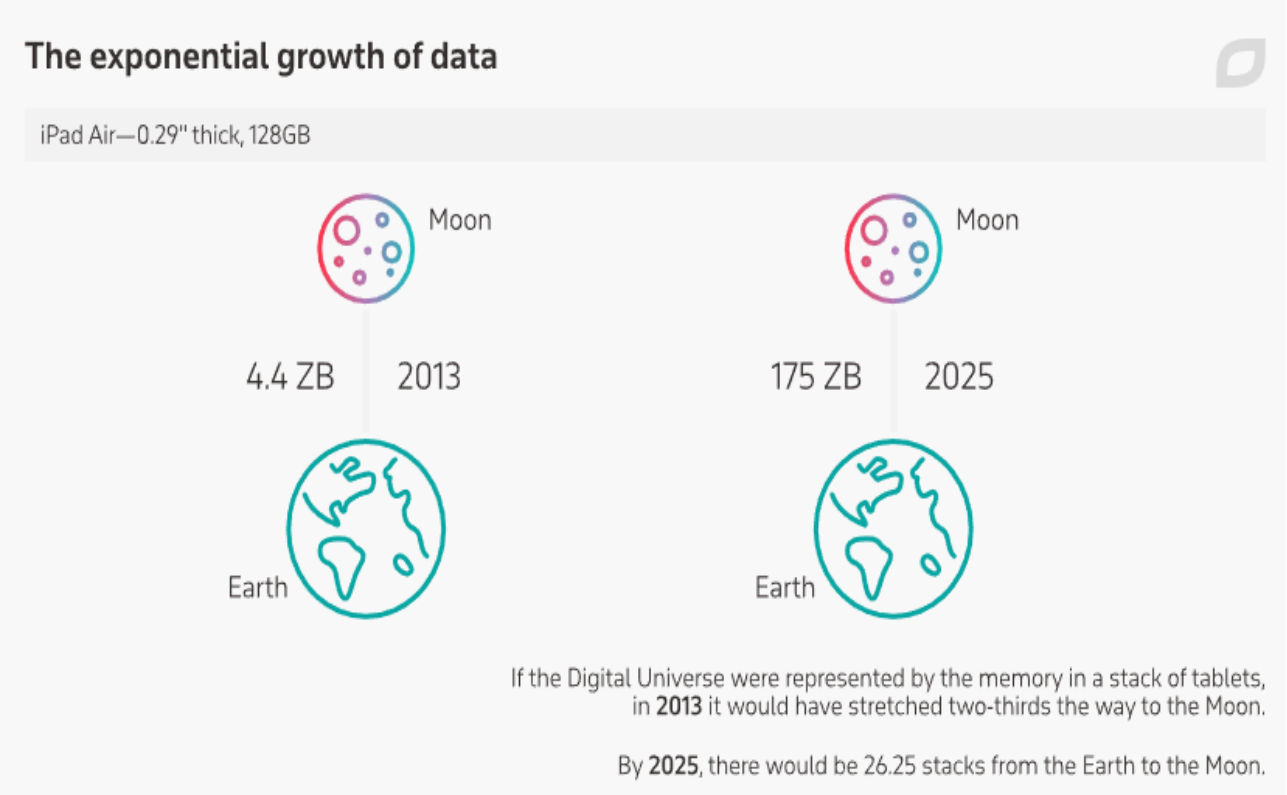


Figure 1.2: Memory if represented as a stack of tablets, the height of that would be 26.25 times the distance between the earth and the moon.

was the beginning of the new era of data expansion. So much so that 90% of all data ever generated in the history of mankind was done in less than the last 24 months. [17].

The worldwide data growth is slated to reach 175 Zetta Bytes by 2025 and that is a pessimistic expectation at best.

This data is conservative, as white papers across the spectrum [18] and [19] have pointed towards the new age of *Artificial Intelligence* that guzzles data at peta byte scale, on a daily basis. Along with modern improvements in surveillance, security, cryptocurrencies, cyber warfare and inter governmental collaborations in data sharing, the ability to process data and generate human readable reports will become quintessential second nature for all modern development.

Even at the accelerated rate, executives have been repeatedly pointing out to the fact that the companies [12] have only temporarily stood up to the challenges in hand, and it could quickly become seemingly impossible to resist the growth of data.

A deeper look at the above studies will show that the pre-pandemic holds on the budgets for data security that had been a barrier to offering almost all products to the customers in a digitized manner have been released and that means, that the systems will largely adhere to these new models and stay sticky. The impact of that explosion has not yet been factored into most of the models that have been presented but have been acknowledged.

Another major factor that will become a contributing factor in the coming years will be the growth in two largely untapped markets of South and South East Asia and the African continent. As digitization, cheap data rates, and increased network connectivity will bring new solutions for the two continents, there will be further expansion in the requirements in the speed of data processing.

Another big piece of the data puzzle is that the data once consumed is massaged, modified and extracted and converted into various different formats all across the board. The same set of customer data is sent to the marketing team for their own correlations, the executive board gets an aggregated and summarized view of the data. The data analytics team has a different view of the data. Normalized data is also sent to the forecasting department of the company. This means, that data is replicated, once it has been created. This means, that the proliferation of data will have another growth dimension that will accelerate over the years, as companies will increasingly move towards adopting Big Data solutions and Artificial Intelligence to drive their businesses forward [17].

1.1 Big Data

With that introduction in our hind sight, let us try to categorically define Big Data. Now that we know, that it is inevitable, it is time to tame the same.

Before we define the concept of Big Data, there needs to be a clear understanding how businesses can benefit from it.

- Applying analytics beyond the traditional data driven decision channels, to support real-time decision making
- Tap into all types of information that can help in this process of decision making
- Empower all people in all types of roles, whether the Data Analyst, or the finance departments, or executives.
- Optimize all the decisions in the system. whether made by individuals or made by automated systems all across the board.

- Provide meaningful insights from all perspectives available in command.
- Provide historic reporting along with real-time analysis.
- Improve Business outcomes and manage risk, both in the current timeframe and at a point in future.

It has now become common practice to differentiate other IT solutions from Big Data by taking into account the following dimensions of it.

- **Volume**

  Big Data solutions will have to manage and process larger amounts of data, when compared to all other technical solutions that exist

- **Velocity**

  Big Data solutions will have to process data rapidly as it comes.

- **Variety**

  Format of the data should not matter, whether structured or unstructured. This means, that *NoSQL* based formats and *SQL* formats will have equal proliferation in the Big Data solutions. This definition also encompasses nested, non nested, memory optimized data formats or just plain text files in the comma separated or tab separated versions as well.

- **Veracity**

  Big Data solutions should also be able to validate the correctness of a large amount of rapidly arriving data.

Amongst the four dimensions as mentioned in the section above, our research work revolves around handling the three dimensions of **Volume**, **Velocity** and **Veracity**.

1.1.1 Volume

When dealing with volume, one of the biggest problems in the system is the lack of visibility into the distribution and structure of the data. For distributed systems, it is far more challenging to compute aggregates and rollups in advance, as they have limited access to the nature of the data lying on the disk. The increasing volumes, also puts constraints on how much of it can be processed at the same time. Most of the analytical models needs sorted data and that needs operations that are size of data. The idea of *Redundant Array of Independent Disks(RAID)* was common in the yesteryears, but the current volumes cannot be sustainably worked on when multiple slow disks are involved.

Organizations are dealing with more stringent requirements for the *Mean Time Between Failures (MTBF)*. Super scalar architectures with millions of cores, have a MTBF of one hour. This means, that systems need to have resiliency in built.

Another important aspect is that a large amount of data is not of any interest. They can be safely compressed with agreed upon loss of data gurantees. This means, that higher compression ratios can be achieved if the customer is willing to tolerate errors for the data that is not of significant interest.

Another important thing to note is that, schemas are not designed very pragmatically, when it is done at scale of the modern era. That means, that data has redundancy built it, there is lack of normalization at source. This adds additional levels of complexity on the existing systems.

An added big challenge in this is that parallel systems can not optimally use and consume this data the way it was expected. The expectation was that the handling capacity will be sub-linear, but the parallel systems keep getting mired in other synchronization and load balancing problems, which are hard problems to solve.

Distributed systems showed a lot of promise in the past, but with the volumes increasing, the single control node answering queries could never scale up as it did not have the latest and greatest meta data about the data, as the union of information was always going to be lossy at the scale.

As repeatedly pointed in the past [17], that volume will keep growing and the above problems will persist. This also brings into limelight, an opportunity. An opportunity to handle data in a way that we have a summarized view of that humongous piece of data lying in the wild in cross continental data centers. If only we could pull in the information without having to pay for the network cost, without having to sort that data, and also guarantee quality data, this could all be solved.

Here comes the basic tenet of the thesis, where we have identified that precise and near accurate summaries of data if maintained can always give a concise idea about the data in the wild. What if that summary could be built in a streaming fashion when that data was ingested ? What if that summary could be losslessly merged in a node downstream.

The solution is to create a streaming solution for the data in the wild which will create small manageable chunks of information that is accurate and can be pulled to one central node without having to deal with petabytes of network traffic. The benefits of that is manifold:

1. All query answering services will need a query optimizer to create a working plan for how the query will be executed. They can make a better distributed or serial plan only if they have some statistics of the attributes involved. They can make a better join order if the statistics are accurate. The margin for error here is very low, as small shifts in the cardinality estimations can cause a tectonic shift in the execution times of those queries.

2. If the statistics can be merged, then work can be done on partial data. Dashboards can still be created if the archived data for the past twelve months have to be built, but one chart cannot fill up the *November* data for some reason. The query can still be built on the merged statistics for the rest of the months and a "close to optimal" solution can be built.

3. Some of the answers can be provided directly from the statistics, if the user is ready to consume data faster, but with some degree of inaccuracy due to the lossy nature of any statistics. This is called **Approximate Query Processing**. Statistics built in the fashion as described in this research, will allow answers to be approximate within error margins with some degree of confidence.

4. Data visualizations can be now be done in a staggered fashion. Early results can be provided from the statistics blobs directly, as they are available in the central computing nodes, whereas the larger data that will take time to process will flow in later, and as and when that actual data is available, the system will refresh the charts to show them as appropriate.

1.1.2 Velocity

[58] Moore's law that famously expected the compute powers to double every 18-24 months, is a thing of the past now. Industry has moved ahead and is figuring ways to handle this stall in the past decade. There has been no significant increase in compute power.

Memory improvements, better network infrastructure using **Software Defined Networks**, optimized network packet management and distributed systems have tried to make as much dent to the lack of superior computing power. There are traditional problems that internodal parallelism cannot solve. This would mean, that the intranode parallelism, due the increasing memory channels , the multi socket and multi tile architectures [34] introduced further levels of complexity that are harder to solve.

This is where we introduce a statistics building apparatus, that does not need the data to be sorted. Peta Byte scale data would need humongous amounts of memory or buffering when sorting data that does not have a traditional index. We have also mentioned that Big Data will also take into consideration non relational data, and unstructured, unindexed data. Without an index, the sorting process will all have to be done by reading database pages on the master node, sorting them(with a possibility of spilling that data to the disk back when building buffers) and then creating statistics. This could be even more painful, in a distributed system where each node could be multiple Tera Bytes in scale, and they have no visibility into what the other nodes have in terms of data. This isolated view of the data means, that the only way to generate that data will be to bring it all into one single space and then computing the statistics out of it.

This is a non-starter for most systems and they need to have a streaming solution that can generate accurate statistics without loss of data. Our algorithms **Q-Regression** and **AuPASS** solve these problems in tandem.

1.1.3 Veracity

Statistics is an approximation of the actual data. With accurate statistics, it allows systems to quickly realize the distribution of the data, or the type of data. This ability is quintessential in the modern data warehouse and datalake architectures. Our research helps by quickly summarizing data in a resource efficient manner.

There are numerous ways to summarize data. To understand how we help, we need to take a quick look at what other methods have been used in the past:

Types of statistics used in Databases

Variable Range Histograms or equi-depth histograms are discussed in [38] and [48]. Serial Histograms are discussed in [36] and [35]. They discuss the merits of *Serial and End-biased histograms*. The performance in terms of the normalized errors has been discussed in terms of number of buckets and the analyzed skew. The main focus of the paper is to be able to create optimal histograms that allow the optimizer to focus on the most frequent elements. Experimentation showed that the onus on the assumption that there is a uniform frequency estimation caused weak histograms. These histograms did not have any order correlation and attribute values with common frequencies were stored in the same bucket. This was optimal for equality predicates and thus needed higher storage since all the values need to be stored in the buckets for optimality. End-biased histograms store the high frequency and

lowest frequency elements in individual buckets and thus adds to the storage requirements for the histogram. The computation also involves a larger set of decision problems. They also suffer from the problem of lack of correlation among elements. Variable-count or equi-width histograms are discussed in [38] and [48]. They are easier to maintain than equi-depth histograms. The variance within the buckets is higher in its case. Equi-depth histograms work well with range queries when the data is skewed. There are assumptions about uniform distributions of frequencies which produce estimation errors for the non-frequent terms. V-Optimal histograms are described in [37]. V-Optimal histograms are computationally intensive. Since the solution involves quadratic complexity, for larger workloads and systems involving thousands of databases they have shown a tendency to slow down the servers hosting the RDBMS system. Many optimizers that prefer not to produce a plan based on default densities or assumed densities may need to wait until statistics are generated for the attribute in contention. There were cases where in-memory statistics needed to be generated for correlations in case of join queries and predicates, where the computation time for a query involving multiway joins and predicates became computationally non-trivial due to the time it took to generate a V-Optimal histogram. Quantile Based histograms are discussed in [59]. [40] mainly worked on large scale distributed systems like Hadoop and thus introduced another performance indicator in terms of communication cost. Another interesting algorithm discussing the possibility of creating histograms in big data systems as a part of the movement of data was discussed in [39]. The paper discussed the idea of using a co-processor like FPGA to offload the computation work. The FPGA accelerator analyzes the tables as they are transmitted from storage to the processing unit. These provides a mechanism to create histograms on the data from the data path. Wavelet-based histograms

have been discussed in [55]. The process involves a multi-resolution wavelet decomposition for building histograms on the underlying data.The histogram stored the cumulative data distribution and optimized space usage. The paper also attempted to offer approximate solutions to user queries. Q-error was introduced in [57]. The paper [42] had done extensive analysis on improving histograms based on the *q-error*. Recent work as mentioned in [73] and [45] create learned models from different query plans that needs to be executed in order to train the models. There have been attempts to learn from data and not from queries, as described in DeepDb [32]. This paper uses the concept of *Relational Sum Product Networks* which are built on top of wavelets or other synopses. Interestingly, the QRegression based synopses can replace those at the leaf level. The training time for the Sum Product Networks is quite large but the updates are faster.

Another paper of interest is *NARU* [78], where offline training to extract multicolumn partial probabilities have been approached. It trains a deep autoregressive model and is trained in a unsupervised fashion. For extremely large relations and joint cardinality estimates, highly accurate QRegression based histograms can supply the base information to train on, thus adding inherent representative sampling.

Amongst them, we preferred to choose *histograms* as our chosen means of representing the data. We implemented **Q-Regression** algorithm to create histograms that are more accurate than the state of the art as mentioned above. This allows for various approximate query processing solutions for real world big data setups, which can utilize the statistics directly to answer queries.

1.2 Can Statistics help in improving energy efficiency?

An interesting treatise on the impact of a bad query optimization can see in this paper [22]. Statistics was a big component that contributed to the decline in performance of a fairly good query optimizer.

In the post pandemic era of global supply chain management issues and continuous threat of prevailing war, energy is of massive geo-political significance. Governments are increasingly wakening up to the realities of unchecked energy usage, better utilization of the existing energy sources and a futuristic plan to shift towards carbon neutrality. With the advent of the Data Centers, a central hub for the energy consumption has evolved and that increases the focus on individual processes that are inefficient.

Studies [76] have proven that certain language paradigms are inefficient by design and that companies are now building solutions that work around this energy gloom.

Although scientists have been pursuing the singular challenge of trying to convince corporations and governments to move towards greener energy sources, counter measures that include reduction in energy consumption by electronic and computational systems is also critical. Not only should the future hold the key to better energy sources, there should be conscious attempt to use that energy efficiently.

The European Commission(EC), created a Code of Conduct for Energy Efficiency [3] in Data Centers back in 2008. Their main aim in this process was to become energy efficient in a manner that it is not disruptive, and businesses can still survive. Like the European Commission, the United States Department of Energy has created a program called the

ENERGY START [9] which even provides guidance on energy efficiency in buildings inclusive of Data Centers.

Energy reduction has become a critical and urgent issue for the database community. A lot of initiatives have been launched on energy-efficiency for intensive-workload computation covering individual hardware components, system software and applications.

A significant portion of compute is utilized by systems that are harnessing the power of data, either to learn from it, or to make knowledge inference systems from the data.

This computation is mainly ensured by query optimizers. Their current versions minimize inputs/outputs operations and try to exploit RAM as much as possible, by ignoring energy. Please refer to 1.3 to see a distribution of the energy consumption components of generic systems.

A couple of studies proposed the integration of energy requirements into query optimizers can be classified into hardware and software solutions. Several researchers have the idea that the operating systems and firmware manage energy and put software solutions in the second plan. This does not distinguish between tasks of operating systems and DBMSs.

Through our work on accurate and faster statistics, we are able to prove that a non-trivial amount of energy is saved when optimizers are well equipped with statistics and can increase the throughput of any given system.

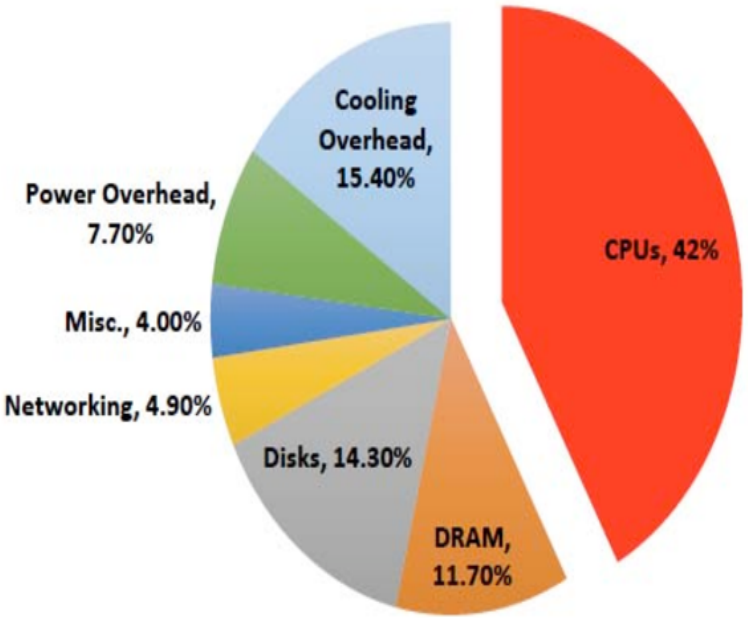


Figure 1.3: Energy distribution across various components involved in Query Processing.

Chapter 2

Problem Statement

We want to start the discussion with a quote from the following article [16]. "Data is becoming the new raw material of business: an economic input almost on a par with capital and labor." This is the defining, watershed moment in history, where state craft will delve in and around data infrastructure and data sharing. Recently, vaccination programs were streamlined and co-ordinated on a global scale and war footing, only due the the data sharing capabilities that exist.

While the larger aspects of Big Data has been discussed, there needs to be a qualified understanding of how that data is used. Data in the store, has no meaning unless users are able to query that. Querying happens through human legible query languages like **SQL**. Over the years a number of standards for the **SQL** language has been proposed. The basic tenets of the domain has remained similar over the last 5 decades of database research. This allows for a good scope definition for most of the problems related to query answering, as most systems have their first principles aligned.

SQL is a imperative language. While it asks the system to provide a result, it does not tell the system how to do that. For example, in other languages like **C, C++, Java, C#**, the reading, parsing, loops and conditional statements will have to be written by the user in a way that the user intends. Performance is up to the ability of the developer of the system to write functional and optimal code. The same is not the case with **SQL**, which expects

the system to find the most optimal way to execute the query and leaves the fundamental resource optimization decisions to the system.

A typical query engine will look like the figure here 2.1. The intelligence is built into the section of the system called the *Query Evaluation Engine*.

And within the *Query Evaluation Engine* lies the brain of the operation or the *Optimizer*. As seen in the 2.2, the *Optimizer*s in most **DBMS** systems follow the **SystemR** [2] model for optimization, in a top-down approach. For the purpose of generality, this thesis uses the same model of optimization for the rest of the document.

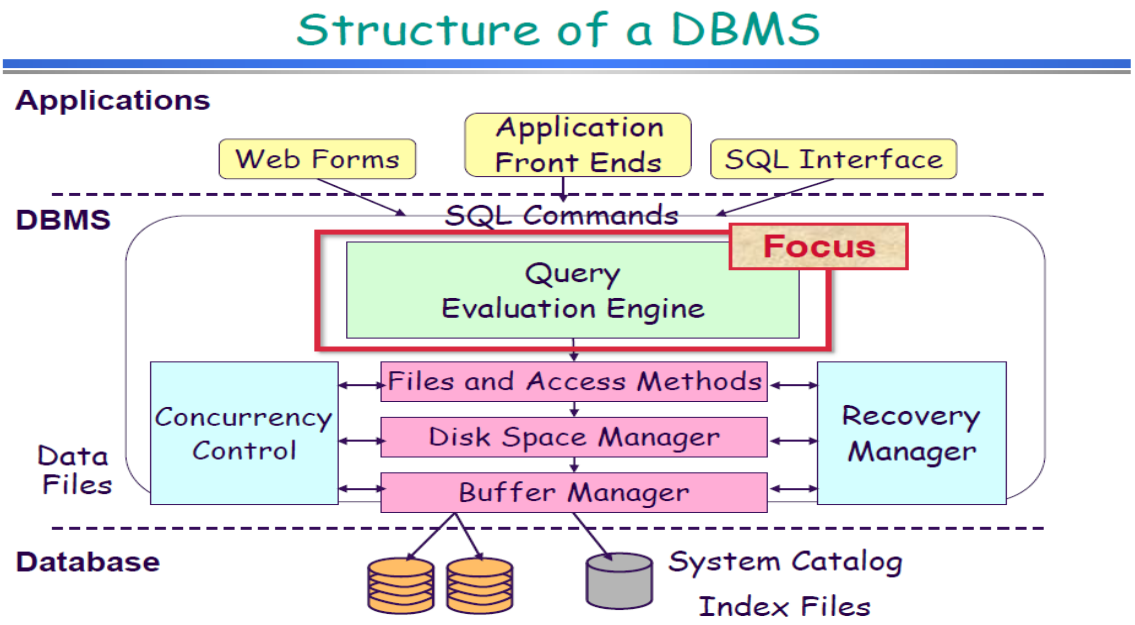


Figure 2.1: Query Engine internals. Figure above shows all the sub-system involved in the process of query execution.

2.1 Research Questions

2.1.1 Optimization Hurdles

A systematic study of various Query Optimizers in accepted production systems has always been a scope of study for many academics in this field [30], [26].

On a standard query engine, irrespective of the top-down or the bottom-up approach of optimization, the optimizer will need estimates of the expected cardinalities of various predicates or joins to create a plan with the optimal join order or the right amount of resources to allocate or to figure out the right index to choose. It also takes calls on whether the number of rows needs a parallel or serial plan based on the parameters it has identified in the process.

The cardinality estimation engine in most standard systems rely on statistics. This can lead to significantly different outcomes due to the difference that exists between the estimated execution parameters and the final actual ones.

The fact that accurate statistics is still relevant in modern engines that have outgrown the original monolithic needs is shown in [43]. The paper cites the relevance of cost based optimizations even in systems that use **SparkSQL** [43]. Companies like **Facebook** have developed their own distributed database engines like **Presto** [70], which moves the computation all the way down to the nodes where the data exists. This was the basic premise of *Hadoop* as well. The paper [43] proves that all the above systems will rely on cost based optimizations to carry out the local operations in a efficient way, within the limitations of the current search space algorithm. The **Hive** meta store is dedicated towards maintaining metadata or statistics of the relations that are used in the system [43].

The relevance of statistics even in polystore and multi-engine systems like **MuSQLe** [28] and **BigDAWG** [23] have been shown to be of quintessential.

A generic discussion for the issues that are plaguing the systems that are used to do big data analysis have been published in [46]. The paper does a deep dive into a comparative analysis of tools and systems that are used in big data processing. There are two approaches to big data processing.

- Batch processing

  This needs pre-processing on the given data as well needs metadata about the data that it is working on. It needs previous knowledge of the table or relation being queried. It is not real time and happens in the background after the data has been ingested. Batch processing is used even in the case of ingestion, cleaning and normalization of

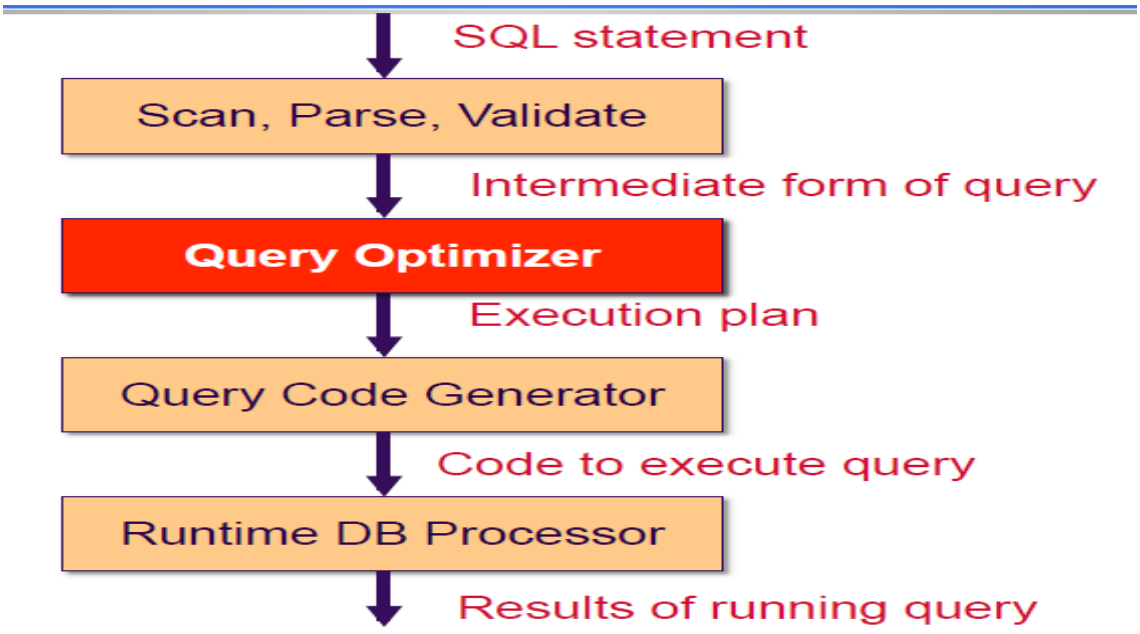


Figure 2.2: The Query Evaluation engine as mentioned in the 2.1 above, can be further divided in to the four sections. The most important sub-system being that of the Optimizer.

data. These systems are good for post processing and warehouse analytics. The data that it works on is not the latest.

- Stream processing

  Batch data processing is not sufficient when it comes to analysing real-time application scenarios. Most of the data generated in a real-time stream needs real time analysis. In [20] and [75], numerous examples have been cited on how most data collected is not converted into meaningful insights as the current limitations due to proliferation of unstructured and non-homogeneous data cannot be queried optimally.

  Stream processing has other requirements like low latency and smaller memory footprint. If a query system tries to answer user queries in batch processing, it will not have statistics about the latest data and will also be working on stale data. This does not work for systems that needs to take immediate action like raising an alert or take mitigation steps to avert a catastrophic situation on the system.

2.1.2 How do we solve these?

Here we introduce the two ideas that form the basis for this research. First is the idea of **QRegression** and **QRegrArea** that tries to achieve better histograms by reducing the error footprint based on the workload and thus learns the best possible bucket partitioning in case of batch processing.

Next we introduce a solution for stream processing, called **AUPASS**, that is a single pass algorithm that does not need sorted data for it to work. It can work on data at

real time using basic sketches like HyperLogLog [25], Count Min Sketch [14] and KLL algorithms [31].

2.2 Optimization Pitfalls

Now that a introduction on the optimizer and other relevant portions of the engine and the importance of statistics has been described, it is time to focus on where the problems can emerge if the statistics is of poor quality. We will take into consideration two different plans for a same exact query, one under the influence of fairly good statistics and the other where the quality of statistics is not up to the mark.

The first plan is the one which uses the "not so good statistics" to generate its cardinality estimates, that drives the final query plan as seen in the image below. Since the estimates provided by the statistics was that of 3160 tuples, the optimizer built a plan with forced parallelism in it. Parallelism is a tool that has to be used with caution. In the hands of a amateur execution engine, it could have devastating affects like stalled queries and reduced throughput, leading all the way till system failures and database unavailability.

This has direct relationship to the business that rely on the databases systems to be transactionally correct. The effect of one bad plan in the perspective of a single query does not always give a tangible idea of the severity of the situation, but when one looks at actual production systems that operate at $\tilde{1}$000s of queries per second, a stall of 500ms could end up with a convoy effect on the whole system 2.3.

The first plan also suffers from the usage of a Index scan which was picked up by the optimizer as it expected a large number of tuples to have flown out of the storage layer. This will have a significant outcome on the final result as a seek could have been utilized at this

point but since the optimizer had bad data to work with, it decided to use the sub-optimal solutions.

The other big problem with this plan is the use of the resource heavy usage of a *Hash Match* that has the adverse affect of pushing a large startup cost. Also this is an operator that is traditionally called a *stop and go* operator, which means it is not a streaming operator and needs all the tuples to be available for it to start providing results upstream. These are some of the severe shortcomings of this plan.

The same problems have been rectified when we have the fairly good statistics that has been used. Here we noticed that the costliest operator in the previous plan has been optimized, which creates the biggest return on investment in terms of improvements. Also the nested loop join will be faster in smaller cardinality estimates like the above.

The resultant plan would need only 5% of the CPU resources as that of the bad plan. This creates a large opprotunity for the system to take on more throughput.

The system shown here is running a **Microsoft SQL Server 2016** and the visualizations are from the **SQL Server Management Studio**.

2.2.1 Whats Next

Until here, we have taken a long journey into understanding **Big Data**, **Query Engine**, **Importance of Optimizers** and the **Importance of Statistics**. At this point, we introduce the next part of the journey, where we technically take a look into the algorithms proposed and their performance.

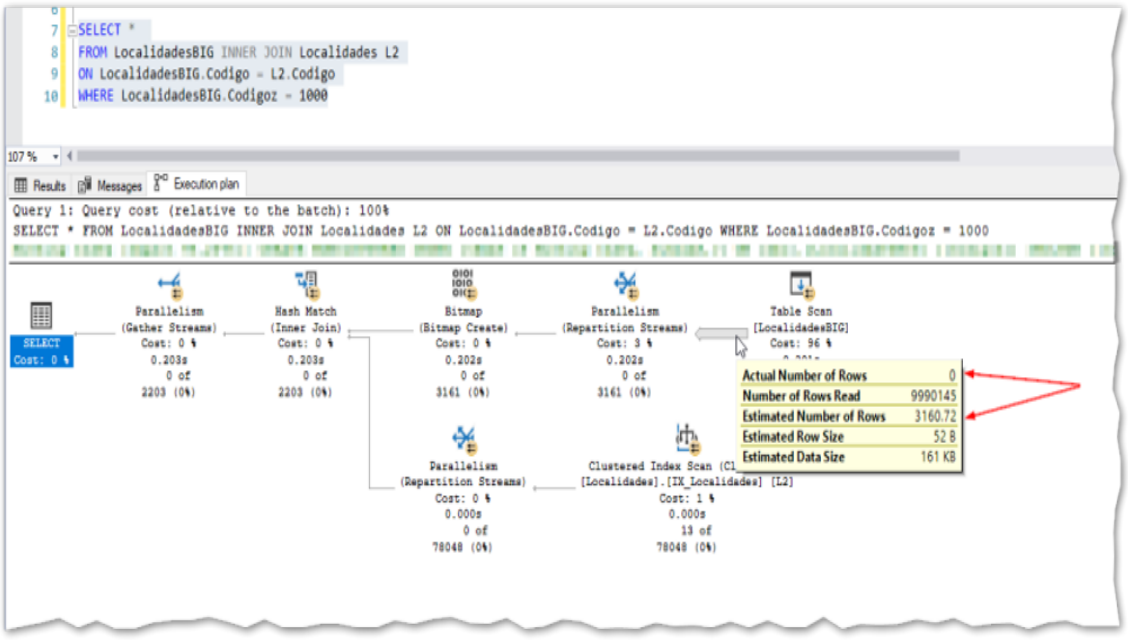


Figure 2.3: When the optimizer is at the mercy of a sub-optimal statistics sub-system, the resultant plan can have severe downsides as seen in the image above.

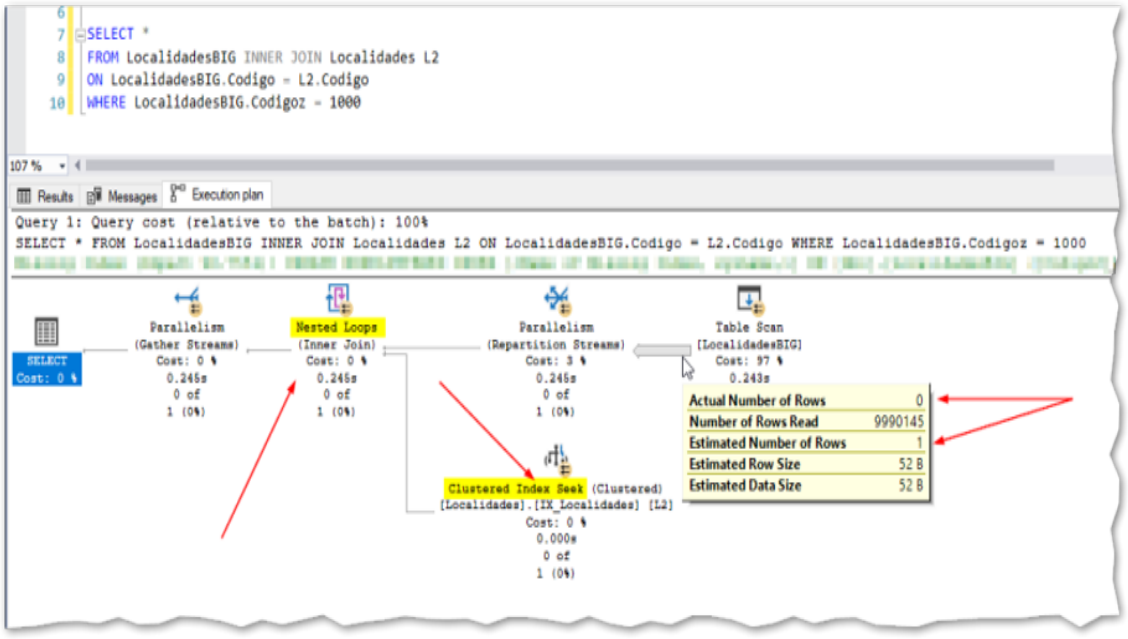


Figure 2.4: In the presence of a fairly good statistical sub system, the optimizer was able to produce the right plan with runtime improvements of 10x.

Chapter 3

Q-Regression, Q-RegrArea algorithms

In this chapter, we introduce the reader to the concepts of our first algorithm that is built to provide near precise statistics with a low memory footprint.

3.1 QRegression based statistics generation

3.1.1 Introduction to Q-regression

For the database to generate an optimal plan for structured queries on the relations and attributes in the database, an optimizer must rely on the cardinalities assumed or deduced from inferred statistics about the attribute that is pre-calculated [69]. These compressed data distributions are called histograms. Wrong estimates could lead to a sub-optimal tree ordering and this has the possibility of exponentially increasing the cost of the overall join tree at execution or runtime. In the case of predicate or a conjunction or disjunction of attribute expressions, the RDBMS system will have to create in-memory, transient distribution of the results by interpreting multiple statistics of the contributing attributes. These statistics are used heavily to estimate the number of rows that will result out of a join.

The optimal join order of a tree of joins could have $\frac{^{2n}C_n}{n+1}$ possible orderings. This itself is an NP-Complete problem to solve, relying or pivoting all its decisions on the statistics that the optimizer can use from the histograms created *a priori*. Statistics about an attribute

will always be an approximation of the actual data and due to memory constraints and the algorithm chosen to create the it, the statistics representation has a smoother representation of the data. This produces a scope for estimation errors when cardinalities are based on the inferred distribution.

A number of papers in the past have tried to create the most optimal histogram under constraints of memory and complexity. There is literature that supports homogeneous and heterogeneous formats of histograms with space efficiency in mind. V-Optimal histograms on the other hand have an exponential challenge in terms of formulating the best bucket structure and boundary placement. There are histograms with equi-depth, equi-width buckets that have tried to solve the problems in the past.

3.1.2 Limitations of the State of the Art

Most RDBMS systems are analyzed based on workload types and there might be a workload that is write heavy and marginal on its reads. Warehouse systems generally tend to be read heavy with data being loaded occasionally. These variations in the workloads generally mean that singular metrics are too generic to be of any material value [33][50].

In [57], a metric called q-error was introduced. This measures the deviation of the estimate in terms of actual as a ratio and finds the maximum possible error from a possible set of range queries, point lookups and number of distinct attributes. [57] points out how by limiting the deviation error to q, the overall cost of a simple join could increase by a factor of q^4. The implications could be that a sub-second query execution time with a nominal error could take days to complete. The importance of accurate size estimations has been underlined by investigations undertaken [65] where it was shown that commercial database

systems are very sensitive to slight changes in selectivity. Open-source and commercial systems have been shown to produce up to 10^4 - 10^8x estimation errors on multi-attribute queries [52].

It is impossible to design a histogram that will be able to answer all approximations within the desired error bounds, hence limiting the worst possible error to q makes sense for theoretical error bounds [44]. A typical database which would have n relations with d attributes, and a workload that at any point of time is compiling x queries that need estimations on predicate selectivity or joins dependent on f attributes, will need $n*f$ histogram objects in memory and $n*d$ histogram objects persisted on disk.

Another attempt to diversify bucket structures to capture more information based on the distribution of the data has been attempted by [42]. For immutable data, where the histogram will be generated once, and will not be modified later on, this is a viable solution. The paper also looks at dynamic programming approaches to achieve lower bounds on the q-error. For data under modification or transactional work loads that needs to have the histograms updated, diversified bucket structures will need additional code complexity. They called these histograms as *Q-optimal histograms* and attempt to achieve the least space requirements amongst other histogram generation techniques. To solve the problem of time complexity of the proposed q-optimal histograms, the paper did provide a heuristics based modified approach that does not guarantee q-optimality.

The observation from the above is that there is no better alternative other than trying to reduce the q-error using custom algorithms. The only caveat here being that the highest error of estimate may be over simplifying the histogram quality and may be rejecting a better histogram for a one that has a overall control on the max error function. This asks for a

Table 3.1: Table showing the distribution of multiplicative errors coming from two histograms ({HistA}, {HistB}). M.E. stands for the multiplicative error of the estimate. If the ratio of the estimate over the actual is less than 1, then it is normalized by inverting it. Bold values show the corresponding q-error.

Query	Actual	Est(HistA)	Est(HistB)	M.E.(HistA)	M.E.(HistB)
1	10	10	10	1	1
2	20	11	15	1.81	1.33
3	30	12	25	2.5	1.2
4	40	13	35	3.07	1.14
5	50	14	45	3.57	1.11
6	60	15	55	4	1.09
7	70	32	65	2.18	1.07
8	80	13	75	**6.15**	1.067
9	90	24	85	3.75	1.058
10	100	35	10	2.85	**10**

better solution that tries to reduce the overall complexity of the process and also tries to fit an optimal histogram for the general case. In this paper, we are trying to improve on the idea of using *q-error* but try and reduce the overall slope of the errors in a way that the memory footprint is satisfied and average error is reduced.

Figure 3.1 shows the distribution of the estimates from the two histograms (*HistA*, *HistB*). The error column describes the multiplicative error derived from the histograms A and B. Figure 3.1 shows the trend line for the multiplicative errors after they are sorted. The figure provides a concrete example of an error distribution where the *q-error* for Histogram A will cause the system to reject it, although it performs better than Histogram B in more than 90% of the queries.

3.1.3 Our Proposed Metric

As discussed above, the inconsistency in quality definition translates to false positives in selection of histograms and leads us to investigate on another metric built on the intercept and slope of the linear regression of the sorted multiplicative errors.

There are multiple benefits to the above defined metric that we are proposing. The first benefit is ability to define the slope of the errors and quick visualization of the median errors in an approximate manner. We will not be tying out results to the extremal or anomalous cases. Optimal histograms can be generated by use of the new optimization objective or reduction in the so called slope and intercept. Dynamic programming algorithm as been proposed in the paper in the section IV of this paper that describes the algorithm.

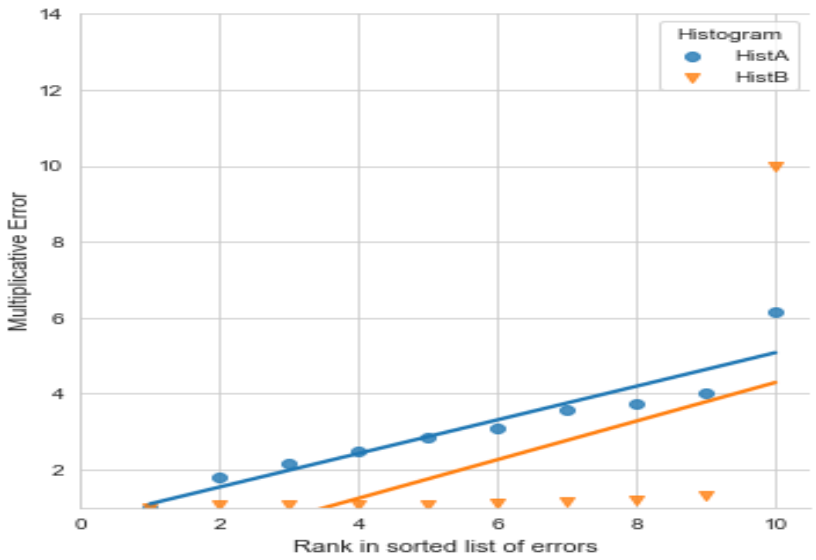


Figure 3.1: Figure showing the distribution of the multiplicative errors for the 2 histograms (*HistA*, *HistB*).Clearly (*HistB*) though superior in more than 90% of the cases, will be discarded as the extremal q-error is higher for it. This could lead to sub-optimal plans for a large number of queries but will in theory, limit the extreme case. The regression lines for the two histograms have been plotted in the same color as the markers.

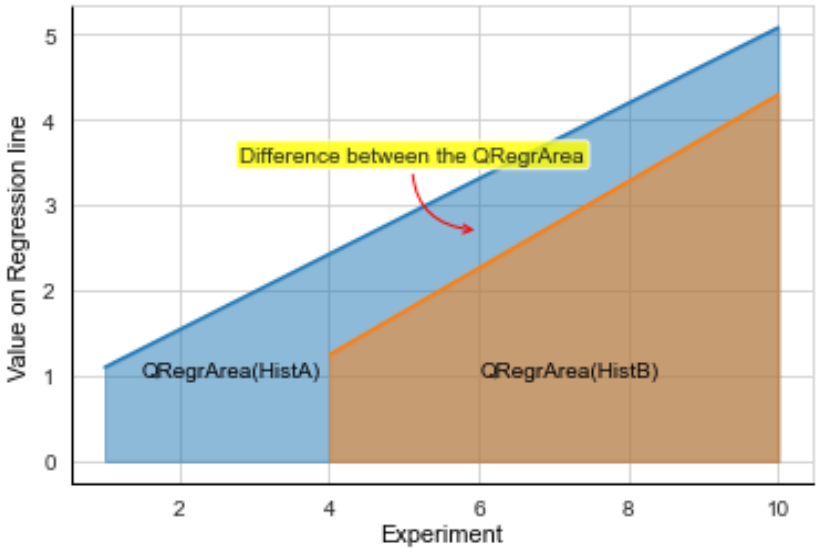


Figure 3.2: Figure showing the area under the regression line(QRegrArea) as a measure of quality of the histogram. As can be seen here, the area QRegrArea(HistB) is much less than the QRegrArea(HistA), and is a better representation of the expected cumulative distribution of the overall multiplicative error using the two histograms, but it will not be selected as it has a higher extremal error.

In order to quantitatively measure the estimated cumulative error of a histogram, we define *QRegrArea*, which is the area under the regression line of a given histogram. Figure 3.2 shows a visual representation of the proposed *QRegrArea*.

In order to optimize the proposed metric *Q-regression*, we develop three dynamic programming-based algorithms.

1. **QHist** - Straightforward implementation of a histogram generation algorithm tied to the piecemeal optimization of individual bucket merge

2. **QHist++** - Improvement on QHist with search for optimal merge boundaries to eliminate possibility of loss of important data points.

3. **QHistComp** - Compaction algorithm to achieve optimal memory and disk space utilization.

We ran comparative experiments against the state of the art and commonly used histograms like *EquiDepth*, *EquiWidth*, *Quantile*, and *V-Optimal* [61].

3.2 Q-Regression and Q-RegrArea algorithms

Here we elaborate on the solutions proposed in this research. The variations in RDBMS workloads generally means that singular metrics are too generic to be of any material value. A lot of discussion about cost implications of the worst case estimations have been discussion in the following papers [33] and [50].

3.2.1 Histogram creation

Before we delve in to the process of creation of the regressions, we would like to discuss the process of creating a histogram from the ground up in a manner that reduces the overall slope β. It is trivial intuition that in the overall sorted order of errors, the estimation function must be completely meaningless if the intercept α is not close to zero for the histogram. A non-trivial intercept for the overall regression means that the histogram is not built on accurate estimations for the frequencies for boundary values of the buckets. All the algorithms under the scanner and our solution rely on the fact that the frequencies captured for the boundary values are accurate and precise. We did not go for a spline-based solution for our problem as that is exponentially more complex to solve.

3.2.2 Notation

We divide the domain of the values in the attribute $\mathcal{A}$ in relation $\mathcal{R}$ into a sequence of m *buckets*, $b_i = [v_{low_i}, v_{high_i})$ where low_i *and* $high_i$ are not the actual values of the attribute but

their indices. These boundaries have the lower boundary included and the upper boundary not included. Some RDBMS systems have the opposite definition of the buckets where the upper boundary is the one that is included and the lower boundary is not included. It is a simple step to modify the below mentioned equations for those situations. If the value set for the $\mathcal{A}$ is $\mathcal{V}$ and the cardinality of $\mathcal{V}$ is n,

$$\forall i \in 1, 2, 3..., m-1 : high_i = low_{i+1}$$

$$low_1 = 1,\ high_m = n + 1$$

3.2.3 Partitioning Problem

As already discussed in [47], the problem of choosing arbitrary partitions with broken splines has already been discussed as that of the *optimal knot placement problem* [8]. This is increasingly complex for non-linear splines. We will be working on linear regressions broken per partition or, in our case, the buckets in the histogram, and we will only be using the actual values that exist in the $\mathcal{A}$ of the value set $\mathcal{V}$. Also, since the $high_i$ values are nothing but low_{i+1}, we can only find the best suitable low_i values for the following set

$$\{low_1, low_2, low_3.....low_m\}$$

This solution relies on a premise from the dynamic game theory called the *principle of optimality* or the *Bellmans Principle*. The intuition here is that, for $l \geq 2 : (low_l, low_{l+1}, ..., low_m) \in \mathcal{V}^{m-l+1}$ is an optimal strategy for partitioning the value range $[v_{low_{l-1}}, v_{high_m})$ using $m - l + 2$ buckets, then $(low_{l+1}, low_{l+2}, ..., low_m) \in \mathcal{V}^{m-l}$ is also an optimal partition of the

$[v_{low_l}, v_{high_m})$ using $m - l + 1$ buckets. For the proof of this theorem please see [47]. When cardinality of $\mathcal{V}$ is very large, the dynamic programming approach will be computationally intensive.

3.2.4 QHist

We introduce the QHist algorithm here which is designed around finding a series of partition boundaries that summarize the local information around the boundaries in a way that reduces loss of granularity within size bounds. We apply a greedy approach to the optimal partitioning problem in such a case. The idea is to start with a large number of small buckets where $card(bucket) << n$ and $card(bucket) \geq 3$. The reason for choosing 3 is to be able to fit a regression through the results and capture the intercept and slope for the errors in each of the buckets. Since the RGE and DCT approximations are the projections of the EMQ approximations in two dimensions, we will solve the approximations for the EMQ for each value in the bucket. If one were to estimate the results for the range queries, the first order would be to estimate the number of tuples whose attribute values fall in the range $[v_a, v_b)$, where $a, b \in \mathcal{R}$ and are actual parameters that are used in the query. There is an interesting analysis of this in [47] Section 5, on result-size estimation for range queries. It involves solving selectivity estimates for more than one dimension.

For each of the trivial buckets, we generate a vector $\vec{b^N}$ which is defined below. Here $\hat{b_i}$ is the actual frequency and b_i is the estimate of the frequency under EMQ for the v_i under the l_q algorithm as mentioned in [57].

The vector $\vec{b^N}$ is a quotient function as defined below, where

$$b_i^N = \begin{cases} b_i/\hat{b_i} & if\ (\hat{b_i}/b_i)\ \leq 1 \\ \\ \hat{b_i}/b_i & if\ (\hat{b_i}/b_i)\ \geq 1 \end{cases}$$

given that $b_i \geq 0, \hat{bi} \geq 0$.

We sort this vector of the multiplicative errors and call this vector $sort(\vec{bn^N})$. We apply a sort on this result set, calling it $sort(\vec{b^N})$. Now the resulting regression line that fits this vector will be an approximation of the errors in the bucket. Any attempt to identify bucket boundaries that reduce the overall slope helps in a multi-pronged fashion. First, it does not need to provide narrow buckets with exact information for some buckets in case of skewed distributions where some zones of the CDF (cumulative distribute function) grow in spurts and the rest of the curve is relatively in plateau or linear. Also, in an attempt to reduce the slope for individual buckets, some of the extremal points that result in high estimation errors do not disqualify the entire histogram in terms of absolute *q-error*. This results in less utilization of memory as buckets with larger spread can be developed and thus fewer buckets are required overall. The experiment section shows the byte size of the histograms. Some researchers have also pointed towards using a mix of different types of buckets in one single solution that have different information packed in them other than the average/cumulative frequency, density and sometimes the frequency of the lower boundary [42].

The line is then calculated using the idea of linear regression as proposed in [62]. We quickly list the algorithm steps for this solution.

algo 1 QHist merge algorithm for building histogram

procedure GREEDYMERGE(V, m) ▷ $\mathcal{V}$ is the sorted vector and m is the desired number of buckets and a is the initial bucket width

  Divide the vector $\mathcal{V}$ in $\frac{n}{a}$ buckets with a EMQ approximations in each

  For the remaining modulo elements, push them to the $(\frac{n}{a} - 1)^{th}$ bucket **for** $i \leftarrow 0$ *to* $(\frac{n}{a} - 1)$ **do**

  end

  Compute new slope β_{merge} intercept α_{merge}

  Insert the newly created $2(\frac{n}{a} - 1)$ pairs to *priority queue* $\mathcal{Q}$

▷ Iterative loop that will create the histogram in one pass **while** *number of buckets* $> m$ **do**

  end

  Merge till we have $\boldsymbol{m}$ buckets

  $c \leftarrow$ *current no of buckets*

  Delete the top element in the $\mathcal{Q}$

  Remove the slope β_{merge} and intercept α_{merge} from the other pair *priority queue* $\mathcal{Q}$

  if $first\ bucket\ =\ True$ **then**

  end

  Create new slope β_{new} and intercept α_{new} for only one case of bucket 0 and 1

  bucket number ≥ 1

  Create new slopes β_{new} and intercepts α_{new} prev, merged and merged, next buckets and insert them into $\mathcal{Q}$

  bucket number $= c$

  Create new slope β_{new} and intercept α_{new} for only one case of bucket *c-1* and c

end procedure

Here follows the process of computing the m values for two adjacent buckets, b_1 and b_2 with their boundary values being $[v_i, v_j)$ and $[v_j, v_k)$. (v_j is the lower boundary for the second bucket b_2 and the upper boundary for the bucket b_1). When we merge their sorted normalized vectors $sort(\vec{b_1^N})$ and $sort(\vec{b_2^N})$, we can improve the computation by using some of the pre-computed values for both the buckets. The sum of the natural numbers and their squares are $O(1)$, constant time operations.

Constant time computation for term $\sum_{l=1}^{l=n} l$ in Equation (4)

$$\sum_{l=1}^{(k-i)} l := \frac{(k-i)(k-i+1)}{2}$$

and the $\sum_{l=1}^{n} l^2$ term in Equation(5)

$$\sum_{l=1}^{(k-i)} l^2 := \frac{(k-i)(k-i+1)(2k-2i+1)}{6}$$

The final value for the *intercept* α_m can thus be computed using the following.

$$\alpha := \frac{(\sum b_{m(1,2)_{i,k}})(\sum ({b_{m(1,2)}}^2)_{i,k}) - (\sum x_{i,k} \sum b_{m(1,2)_{i,k}})}{(k-i)\sum ({b_{m(1,2)}}^2)_{i,k} - \sum {b_{m(1,2)}}^2_{i,k}} \tag{3.1}$$

and the value for *slope* β_m can be computed using

$$\beta := \frac{(k-i)\sum x(b_{m(1,2)_{i,k}}) - (\sum x_{i,k})(\sum b_{m(1,2)_{i,k}})}{(k-i)(\sum ({b_{m(1,2)}}^2)_{i,k}) - \sum {b_{m(1,2)}}^2_{i,k}} \tag{3.2}$$

The sum of the product terms of the index and the error cannot be reused because merge sort of multiplicative errors from two buckets will cause the error term to change as

well as its order in the final vector.

$$\sum_{l=1}^{(k-i)} l(b_{m(1,2)_{i,k}})_l = \sum_{l=1}^{(j-i-1))} l(b_1)_l + \sum_{l=j}^{(k-j-1))} l(b_2)_l$$

3.2.5 QHist++

The merge algorithm currently will always merge 2 buckets of same size into one. Since the algorithm is built agnostic of the size of the two merging buckets, it can be dynamically tweaked to allow for merging on interesting boundaries. In $QHist++$ we reduce the possibility of swallowing major boundaries in the bucket. Algorithm (2) defines the procedure to determine whether 2 buckets can be optimally merged. If they are not, then we will ignore the pair and pop it off the *priority queue* $\mathcal{Q}$ but not adjust the elements in the Bucket.

3.2.6 QHistComp

One of the problems with the regular version of the algorithm is the forced compaction to a predefined size m. Based on the size requirements for most major $RDBMS$ systems, we found the typical size of a histogram to be limited within 3200 bytes to 4000 bytes but that might not be optimal for large data. In case of larger histograms, further compaction is needed. We have created a compression algorithm to merge the buckets if there is no significant loss in the quality, as can be seen in Algorithm (2).

algo 2 Optimal Merge Algorithm for QHist++

procedure IsOptimalMerge(a, b)
    $a_m \leftarrow cardinality\ of\ boundary\ for$ a
    $a_{avg} \leftarrow average\ for\ bucket$ a
    $b_m \leftarrow cardinality\ of\ boundary\ for$ b
    $b_{avg} \leftarrow average\ for\ bucket$ b
    if $a_{avg} \gg b_{avg}$ *or* $b_{avg} \gg a_{avg}$ **then**
    end
    return $False$
    $a_m \gg a_{avg}\ or b_m \gg b_{avg}$
    return $False$
    $\frac{a_{avg}}{b_{avg}} \geq 2\ and\ \frac{a_m}{a_{avg}} \leq 2$ ▷ Find the optimal spot to move the boundary **for** $i \leftarrow 0\ to\ card\ of\ b$ **do**
    end
    If adding value reduces $\frac{a_{avg}}{b_{avg}} \leq 2$ break
    $\frac{b_{avg}}{a_{avg}} \geq\ 2\ and\ \frac{b_m}{b_{avg}} \leq\ 2$ ▷ Find the optimal spot to move the boundary **for** $i \leftarrow 0\ to\ card\ of\ a$ **do**
    end
    If adding one more value from a reduces $\frac{b_{avg}}{a_{avg}} \leq 2$ then break
    return $True$
end procedure

Chapter 4

AuPASS Algorithm

4.1 Motivation for AuPass

A study by Abadi et al [1] has defined the new direction for the database community in recent times. Technological breakthroughs in `machine learning`**(ML)** and `artificial intelligence`**(AI)** and lower barriers to writing ML-based applications using frameworks like TensorFlow and PyTorch, along with architectural innovations in neural networks and specialized hardware both in private and public clouds, means, that the consumption and demand for data is exploding. These innovations have succeeded the two prior phases of columnar storage era and the data analytics era.

As more training models needs relevant data, data storage architectures have seen significant changes like the introduction of *serverless* and *data lake architectures*, which use on demand elastic compute and elastic storage services in cloud. This decoupling of data and storage and the demand to apply query surface for unstructured data in constrained environments means, there is a need for basic query optimization with streaming data.

The emergence of *Industrial Internet of Things* focusing on domains like retail, healthcare, manufacturing has accelerated in the last five years. Availability of cheaper sensor devices, tiered network connectivity across the spectrum and new innovations in the network

stack with optimized bandwidth architectures has changed the face of the data analytics infrastructure.

Data analytics in 2020's scale has now become a multidisciplinary field that impacts decisions across companies, organizations and scientific research. All the data collected today is not converted into insights at the same tenacity.

In [20] and [75], numerous examples have been cited on how most data collected is not converted into meaningful insights as the current limitations due to proliferation of unstructured and non-homogeneous data cannot be queried optimally.

A few important factors that needs to be kept in mind when working with data at scale are the following

- Without metadata about the distribution of data on a particular attribute, it is difficult to arrive at an optimal plan for a given join. Most big data solutions out there, rely on denormalization as a solution and remove the need for any joins. There are severe limitations to that model, as there are a large number of analytical queries with self joins and *common table expressions* which needs meta data or statistics about the attributes in the relation.

- Most statistics generation methodologies on data, need data to be pre-sorted. Variable Range Histograms in [38] and [48], Serial Histograms in [36] and [35], Variable-count or equi-width histograms in [38] and [48], V-Optimal histograms in [37], Quantile Based histograms in [59] and Hadoop specific algorithms as mentioned in [40] need the data in a pre-sorted order. This means, either the data in the data lakes have to be indexed

or the sorting has to happen at the compute layer. For large relations as seen in modern warehouse analytics, these memory requirements are prohibitively large.

Also, achieving high accuracy would also mean that dynamic programming solutions like [37] are applied to the data and they could be compute intensive.

- With most applications at cloud scale and tenant sharing configurations being used to overbook compute, there is a resource constraint on generic systems. Targeted solutions need custom hardware or special relations with the public cloud company to achieve the desired behavior in terms of memory and parallelism.

- Due to the varied nature of the data sources and the non-homogeneity of the source relations, caching all the information is nearly impossible. In a distributed system, there is no guarantee that there will be use for the cached data within a small period of time.

- With *industrial internet of things*(**IOT**), another challenge is the edge computing at individual router, sensor, mini devices, watches or switch level. This would need algorithms that can create statistics on the data in a single pass with low memory footprint.

4.2 Solution Proposed

The most important factors that have been taken care of in our research have been the following

1. Memory footprint of the solution. Since the solution is attempting to solve the problems of Big Data in a distributed environment. This is clearly of significance that the memory footprint is as low.

2. The next most important factor to observe is the requirement that can be worked upon both in a window and window less fashion.

3. The results should be losslessly merge-able. This means that in a distributed environment there should be sketches that are generated at intermediate nodes that can be aggregated at a control node.

The idea is simple and easy to integrate. With sketch based approximation solutions like count min sketch with a reservoir sample, it will be easy to generate all the heavy hitters in a streaming fashion. A KLL [31] sketch, can also be generated in the same pass, so that the solution is a true single pass solution. The KLL Sketch will provide the quantiles that we want to place as the boundaries of the histogram. This will almost always be the heavy hitters that have been identified by the Count Min Sketch algorithm.

At the end of the collection and aggregation of the sketches(which happens at the master or control node), the heavy hitters are applied as the histogram boundaries, based on their nearest quantile values.

This approach allows for multiple simpler problems to be solved within limited memory requirements(as it is a streaming algorithm with the ability to generate sketches in a single pass). The solution creates sketches that are all easily merged at the compute node.

There is no need for sorted data and that reduces both compute and memory requirements. Also, the algorithm does not need to infer or build any apriori knowledge about the system.

The need for accurate statistics is critical for handling queries that handle huge volume of data [69]. Another big problem other than accuracy is the ability to generate statistics in a streaming fashion in real time with one pass algorithms. There have been a few algorithms that have been proposed in the past like the Ben Haim paper [5] and SLIQ by Mehta et al. [56] and SPRINT [71]. They have the requirement that the data be sorted. When the system is dealing with data from multiple sources and streams, sorting is not an option as new data will not be reflected in the data that has been processed. Sorting, with sampling and without sampling, both have a big problem as they need either the data to have an index, or the data needs to be sorted in place. This is a non-trivial problem when the devices have limited memory. An added challenge is that compute intensive statistics generation methods cannot be used on such devices.

Some novel approaches to use sketches have been tried with approximate representations as in [27] and SPIES [41]. They also target a different problem altogether, in decision tree classification problems and not statistics or histogram in general.

We are proposing a new algorithm that is built on top of existing sketches that have been proven to provide a good error margin with high confidence. This does not need windowing and thus can be used to incrementally build on top of the information it already has.

4.2.1 Central Idea

The central idea for this algorithm is that the heavy hitters, the items with the highest frequencies can be used to identify the bucket boundaries. This solves the partitioning problem for histogram generation. For this, a count min sketch is used. For finding the overall density of the distribution, distinct count is found using the HyperLogLog algorithm. The quantile ranges are found using the KLL algorithm. Once the sketches are prepared in a single pass, they need one single pass over the sketches to identify the heavy hitters. These heavy hitters are then correlated with the KLL sketch to identify the nearest quantile to which they map. This allows for more structured histograms which are not end biased.

4.2.2 Algorithms used

The three algorithms that are used in the AUPASS idea are the following.

1. **Count-Min Sketch**

   The Count-Min sketch is a compact summary data structure capable of representing a high-dimensional vector and answering queries on that vector. They are very useful in answering point queries and dot product queries and have good accuracy guarantees [14]. They are heavily used in the industry to identify heavy hitters. The idea of heavy hitters is central to the AUPASS algorithm.

   The data structure is capable of updates and deletions to the data. It is thus useful in a transaction heavy workload as well, where the existing data that has been processed can have modifications done on them and the Count-Min sketch will be able to consume

them as well. It is also capable of working over large streams of updates, at high fidelity and throughput.

The data structure maintains the liner projection of the vector with a number of other random vectors. These vectors are defined by simple hash functions. Typically, universal hashing families are used for such random vectors. Increasing the range of the hash functions will have a direct correlation with the accuracy guaranteed by a particular sketch for a given size. Similarly, the number of hash functions have a direct correlation to accuracy as well. This allows for the CM sketches to be linearly scaled as the requirement increases for the sub-system involved.

2. **HyperLogLog**

HyperLogLog is a count distinct solution, which approximates the number of distinct elements in a multiset. Calculating the exact cardinalities of the unique elements of a multiset requires an amount of memory that is proportional to the cardinality of the relation involved.

This is a typical pain point for big data solutions that are dealing with billions of tuples

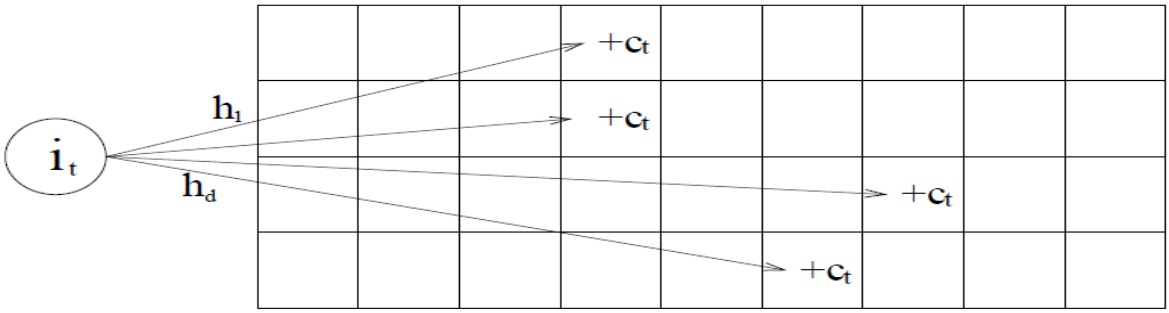


Figure 4.1: Random vectors computed from the source vector in the Count-Min sketch.

and the memory requirements for highly variable data could need gigabytes of memory in the system. Probabilistic cardinality estimators, such as HyperLogLog algorithm used significantly less memory and thus the cost of obtaining a near precise distinct count is reduced by a large factor. HyperLogLog provides the distinct count, that is then used to calculate the density of the attribute under consideration. This helps with point and range queries for the non heavy hitters and thus give a relatively high confidence for the low frequency elements within the relation.

3. KLL The Karlin, Lang and Liberty sketch [31](FOCS 2016), hereby referred to as the KLL was introduced to solve the problem of finding quantiles in a data stream using a small footprint in memory. The per bit accuracy is highly optimal due to the structure of the algorithm that has been advised. The key features of the sketch are as follows

   - The sketch is compact and thus can be used in edge devices.
   - The sketch can be serialized and thus allows for their effective merges in the upstream nodes.
   - The parameter K that affects the accuracy need not be a power of 2

4.3 Algorithm

4.3.1 Preliminaries

A large number of papers have worked on improving the behavior of **Count Min Sketches** to handle low frequency events as they get lost in the process. In text mining, this is a significant problem. To counter that, we will use a larger memory base for our sketches.

Since the sort requirements are taken away from the system, and the algorithm is a single pass solution, the memory saved from not having to create buffers to sort and merge, the increased footprint of the sketches is contained [60]. The original work by Cormode et al [15] is a good reading on the same. Similarly, the **Hyperloglog** algorithm is well described by the work done by P. Flajolet, E. Fusy, O. Gandouet and F. Meunier [25] will give the reader an insight into the same.

The **KLL** algorithm has been a ground breaking algorithm in the approximate percentile arena. Work done by F. Zhao et al [80] describe the remarkable utility of this sketch algorithm that allows for a small memory footprint while keeping a near precise hold on the quantiles and percentiles of a distribution. The algorithm is streaming as well, and fits well into our scheme.

Since there is data loss when it comes to converting the input tuples through the hash functions used in the **Count Min Sketch**, we will need to have a reverse lookup using some form of a probabilistic sampling algorithm, that will be utilized to get the original tuple back. We chose the Reservoir algorithm to do that task for us.

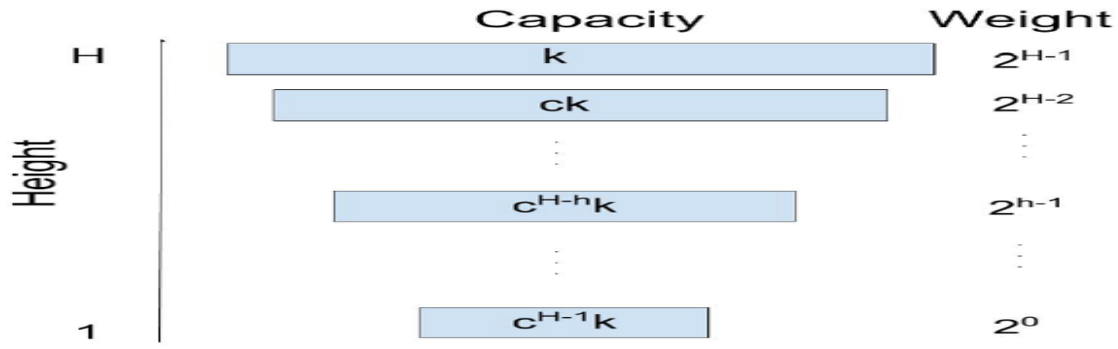


Figure 4.2: Demonstrating the elevation algorithm for KLL

The problem is to select without replacement a random sample of size n from a set of size N.

The first n records of the file are pushed to the reservoir. The rest of the records are processed sequentially. The principle of the Reservoir Algorithm is that at any point in time after the initial hydration of the Sample is done, a size n can be extracted from the current state of the reservoir. This algorithm runs in O(N) because the entire file must be scanned.

Under the assumption that we have a continuous random variable with acceptable uniformity in randomness, we start creating a Reservoir. It can be generated very quickly, in constant time. So, At the ith element of S, the algorithm generates a random number j between 1 and i. If j is less than n, the jth element of the reservoir array is replaced with the ith element of S. In effect, for all i, the i'th element of S has the probability of n/i for being included in the Reservoir.

Assume $i + 1$ to m elements kept in reservoir. Similarly the probability of not being replaced element from reservoir is $1 - (\mathrm{m} - \mathrm{i})/\mathrm{m} = \mathrm{i}/\mathrm{m}$.

Finally the total probability of surviving and not replaced elements is $(\mathrm{n}/\mathrm{i}) * (\mathrm{i}/\mathrm{m}) = \mathrm{n}/\mathrm{m}$.

In the end, we end up with a Reservoir, where each element has uniform probability of being included or not included.

The $Card()$ function used in the algorithms are to find the cardinality of the input.

4.3.2 Algorithm

The **QHist** family of algorithms follow either a greedy approach to reach a hypothetical optimal histogram, whereas the **AuPASS** is streaming algorithm that does not have to work

with existing data. The benefit of this algorithm is that this can piggyback on the ingestion pipeline and thus end up with accurate statistics at the end of a data ingestion process. In the case of partitioned data, a merge algorithm will have to be executed with the existing sketches for the other partitions. The end result will be same as if all the partition data was seen and computed in one go, as the properties of the **Count Min** and **Hyperloglog** sketches make them capable to merging from two different sources without loss of precision of the final sketches. So if the pre-existing partitions had their own sketches, the same can be merged to the newest partition.

Algorithm steps

Since the implementation of the KLL algorithm is quite complex, the addition of those steps here would bloat this algorithm by pages, the reader is asked to read the work by Zhao, Fuheng and Maiya [80] to get the central idea on how the final sketch will look like and how to retrieve elements from the same.

The algorithm for the collection shows the idea of a streaming pass over the data that is getting ingested on the fly and being captured into summaries that will be used to finally merge into a histogram 4.

Another pass of data will be needed to organize this information.

4.3.3 How does AuPASS fit in with QRegression Family

The **AuPASS** algorithm allows for a single pass, streaming algorithm to merge statistics from various distributed nodes. Once the histogram is generated, it will help the system to be performant from the moment the data has arrived. After the initial loading period, for

algo 3 AuPASS Collection Algorithm

1: **procedure** COLLECT COUNT MIN SKETCH AND RESERVOIR BUFFER($S, m, n, hfArray, SketchArray, Reservoir, h$) ▷ S is the streaming vector and m is the desired number of buckets
▷ a is the initial bucket width and n is the desired number of Universal hash functions needed by the Count Min Sketch.
▷ h is the number of buckets in each hash sketch. hfArray is the array of hash functions that belong to the same universal hash function.
▷ SketchArray is the array of the buffers in which the Count min Results will be posted.
Create a buffer of possible heavy hitters using Reservoir
for $i \leftarrow 0$ *to Card(S))* **do**
for $j \leftarrow 0$ *to* n **do**
2: $x \leftarrow hfArray[j](S[i])$;
3: $y \leftarrow x\%\ Sizeof(SketchArray[i])$;
end
4: **if** $(Reservoir.AttemptToEnter(S[i]) == success)$ **then**
5: $Reservoir.Enter(S[i])$;
end
6:
end
7:
8: **procedure** HYPERLOGLOG AND KLL SKETCH($S, hll, hf, HLLSketch, KLLSketch$)
▷ hll will be the number of buckets in the Hyperloglog sketch.
▷ S is the streaming vector. The data need not be sorted.
▷ hll is the number of buckets in the Hyperloglog sketch
▷ hf is the hash function chosen.HLLSketch is the scratch buffer in which the calculations are done.
for $i \leftarrow 0$ *to*$Card(S)$ **do**
▷ For every tuple use the hf function to figure out the bucket that will be updated by using the modulo with the hll bucket count.
▷ For every resultant hash, cache the number of leading '0's in the result in the variable x.
9: $h \leftarrow hf(tuple)$; ▷ If the value contained in x is ¡ than the existing count in the bucket, then that value is replaced with x.
10: $x \leftarrow 0$;
11: $x \leftarrow h\%hll$;
12: $y \leftarrow PrefixCount0(h)$;
if $y \geq HLLSketch[x]$ **then**
13: $HLLSketch[x] \leftarrow y$;
end
end
14:
15:
=0

algo 4 AuPASS Merge Algorithm From Sketches

```
1: procedure MERGE(S, m, SketchArray, Reservoir, HLLSketch, KLLSketch)
   ▷ S is the vector
   ▷ m is the desired number of buckets, SketchArray are the CM Sketches
   ▷ Reservoir is the reservoir sample
   ▷ HLLSketch is the Hyperloglog Sketch
   ▷ KLLSketch is the KLL Sketch
2:     resultArray ← m × 2 empty buckets
3:     currentCount ← 0
   for i ← 0 to Card(S) do
       for j ← 0 to Card(KLLSketch) skip (Card(S)/m) × 0.5 do
4:         x ← KLLSketch[j]
5:         y ← ReverseLookup(Reservoir, SketchArray)
           if CountMinSketch.Lookup(x) ≥ Card(S)/m then
6:             resultArray[currentCount + +] = y
           end
7:
       end
8:
       end
9:
10:        finalBucketCount ← Card(resultArray)
   while finalBucketCount ≥ m do
11:        Merge adjacent buckets with the least amount of data loss.
12:        Use the information from the HLL bucket to count the number of distinct elements.
13:        Use the distinct counts to find the overall densities of values.
       end
14:
```

transactional datasets, where modifications happen frequently, and indices are built over a period of time, the **QRegression** family of algorithms will be suitable when refreshing the statistics for the given attributes that have changed significantly since the beginning of the operations. For cases where an index does not exists and there might be a need to update the statistics, it will still be meaningful to use the **AuPASS** algorithm, with sampling builtin, so that extrapolation of the distinct counts and values are done properly.

AuPASS is also useful for edge nodes, in a sensor based data collection environment like hourly or real time monitoring systems, where errors are rolled up and aggregated all the way to the control nodes that can take action based on the data coming from each individual sensor.

Given that the sketches are all losslessly merge-able, the data in the individual nodes can be kept alive in memory as it takes a small footprint(few kilobytes). These temporary sketches will keep updating themselves as more data enter over time. This provides the system with a robust mechanism to create statistics in more than one ways and also use less resources in the process.

Chapter 5

Experimental results

5.1 Setups Chosen

Let's deep dive into the environment setup for the experiments and the models chosen. The setup is similar for both the **QHist** family of algorithms and the **AuPASS** algorithm. The datasets used are the same as well. Interestingly, the **AuPASS** is used when ingesting the data without any decision metric, it will build statistics for all the columns involved. This leads to interesting bloat in the meta data for systems that will use **AuPASS**. For the purpose of this thesis, we have chosen to only update the statistics of the data types that we are interested in. As a result, all columns of integer, datetime, money, decimal have been included by default. The size implications of this has shown an average increase of 7% disk space consumption on the database end. The statistics are not loaded into memory until needed. For a fully balanced workload that covers most relations and attributes, it will be very helpful for the user to already have access to the statistics on disk. Creating the statistics is a blocking process and this can halt systems or causes a compilation storms if a system is getting warmed up after a server restart.

5.1.1 Choice of Models

We chose five different state of the art models for histogram building and ran experimentation's on their variations across workloads, table sizes, bucket counts and distributions. We have not shown any experimentation's across trivial histograms as discussed in [61].

We reuse some ideas from [61]. A *histogram* on an attribute A is defined as a set of mutually disjoint subsets called *buckets* which approximate the frequencies and values using a common fashion. The algorithms we picked for the experimentation belong to the commonly used partition class of serial histograms and one variation of the V-Optimal algorithm for end-biased histograms (they have all but one of their buckets as singleton). Sort and source parameters are chosen from either the spread (S), the attribute values (V), frequencies (F) and cumulative frequencies (C).

Equi-sum(V,S) also known as EquiWidth

These histograms group contiguous ranges of attribute A values into the buckets with a spread of $(1/m)$ times the spread of the attribute, where m is the number of buckets.

Equi-sum(V,F) also known as EquiDepth

These are like the EquiWidth histograms but the sum of the frequencies of the attributes in each of the m buckets will be same. They are also known as EquiHeight histograms. They have the added challenge of determining the right bucket boundaries in skewed distributions.

V-Optimal(F,F)

This algorithm attempts toward grouping attributes with similar frequencies in their own buckets. This has the advantage of recording heavy hitter attributes with large frequencies in the histogram.

V-Optimal-End-biased(F,F)

This is a form of end-biased histogram where some of the highest frequencies and some of the lowest frequencies are placed in their individual buckets and thus they have a large number of singleton buckets.

Quantile(V,C)

Quantile histograms place the boundaries on discrete ranks of the cumulative frequencies for the attributes.

Spline(V,C)

The algorithm is inspired by efforts in numerical analysis to approximate curves.

The first database system we used is PostgreSQL 12 (called as System X going forward). It has two ways to measure statistic. One is the pg_class, which stores the distributions number of pages and tuples in the relation, and the other is the pg_stats view that works on the table pg_statistics. The format for the statistics is mainly directed towards the most frequent attributes in the relation and its frequency. It does not have any information on the averages of the distribution per bucket but only provides information on the most frequent

elements and their frequencies and the boundary values. It only shows 100 such frequent values but one can change the configuration to show more than the top 100 frequencies.

For our second system we chose the system MySQL version 8.0 (called as System Y henceforth). This relational database management system keeps the histogram information in a json. The solution provides a number of knobs including ability to move the sampling rate between 0 and 1, where 0 means nothing is sampled and the statistics is default and 1 means, that all the rows are sampled. For our case we used 1 for our sampling. The system allows for the histogram to differ between singleton and equi-height solutions. The number of buckets can be specified. For our solution we used 100 buckets as that is the optimal for tables with 1000 distinct values.

We chose MariaDB 10.0.1 for our third solution (called as System Z from here on). This solution stores statistics about the table, the columns and the indices. The histogram typically has 2 different precision systems (single precision and double precision). The precision will change the size of the statistics used by system. On large clusters, the size of the column statistics will be critical for system performance as each relation, with multiple attributes and an exponential growth with indices, the number of statistics could start pushing the performance thresholds. This system allows for the histogram size to be modified between 0-255 buckets.

All the three systems support scalar datatypes and have similar interfaces in terms of programmability. All the systems need external stimulus to create data synopses. They also provide configuration and knobs to tweak the quality of the data. There was another system that was in consideration but that had vastly different information and we refrained from

using it to reduce interference from unknowns. Also, we have tried to reduce the noise from sampling and used full scan for all the statistics generated in the system.

Although it has no implications on the outcome of the results of our experiment, we have tried to keep the histograms for a given attribute within 4000 bytes using the various knobs like additional compression and bucket count.

5.1.2 Query Types

For AEMQ queries, we used all the unique values for the attribute to generate the queries. For cases where the number of distinct values is huge, we have constrained them to 2400 distinct elements. For EMQ queries, we picked all values in the domain `lb` and `ub` and not constrained to the attributes that already exist in the relation. This is representative of the scenario where client systems have no prior knowledge of attribute values in the relation and cannot expect a count for all their queries. For RGE and DCT, we had to select random sampling for the `lb` and `ub`. This has the possibility of missing out on interesting cases but for an attribute with million distinct values, this has the possibility of $\sim$trillion combinations. Random selection of the `lb` and `ub` across multiple experiments gives a higher confidence in our measure as the same queries will be used for systems treated with our novel idea and without.

5.1.3 Datasets

Since the errors for integer data types are easy to compute, we have chosen attributes from various datasets that are of integer type. Various commercial database systems allow for other data types like currency, date and time, spatial, hierarchical, or string but any sortable

datatype can benefit from the idea proposed if the cardinality estimator system relies on the histogram and the information stored in the buckets for such data types. With string, the errors cannot be identified and similarly for non-atomic types like datetime. Rounding errors have introduced complications in the outcome of the system when dealing with lower *q-error* bounds. We have limited to picking integer types to the sets under observation. Although, vectorization of the string data types or usage of Levenshtein's distance [53] could create the opportunity to use this algorithm as well.

We now describe the datasets that we have used. By `quartcenew (qcew)` we mean the 2019 Quarterly Census of Employment and Wages dataset using North American Industry Classification System also called (NAICS). We only considered two numeric attributes with interesting distribution here and they were `taxable _quarterly_wages` and `total_quarterly_wages`. The other dataset used is the `burlabstat (bul)` which represents unemployment data for the state of Alabama from the Bureau of Labor Statistics. We used only one attribute from this dataset called the `labor force`.

We have also introduced a randomly generated data set called `randomdist (rdis)` for introducing the possibility of finding the behavior on absolutely random data. Although, this will never be the case out in the wild, the same treatment done with or our without our idea did show improvement and we wanted to present that fact. Other than these, we have presented results from experimentation done on various distribution types including normal, multinormal, discrete and continuous distributions and different table sizes.

As mentioned in the beginning Section V, we have used 3 datasets for our experimentation, choosing to use traditional histogram construction algorithms compared to our novel approach. We now present data to illustrate the effect of our implementation.

Table 5.1: Distribution of *multiplicative errors* across all Baseline models under consideration, EquiWidth(EW), EquiDepth(ED), V-Optimal(VO) baseline Algorithms. The system that has most number of queries with multiplicative error ≤ 2 and least number of queries ≥ 5 are considered to be the most performant. This means, that for each dataset, higher numbers on the first row is better and lower number on the last line(representing queries that had a multiplicative error more than 5) is better. The winners for the dataset for a particular type of query is marked in bold.

		EW			ED			VO			Q		
		EMQ	DCT	RGE	EMQ	DCT	RGE	EMQ	DCT	RGE	EMQ	DCT	RGE
qcew	≤ 2	80%	73%	72%	81%	76%	74%	91%	**90%**	84%	67%	76%	70%
	>5	07%	07%	15%	06%	09%	08%	03%	04%	09%	02%	04%	05%
bul	≤ 2	83%	72%	73%	88%	71%	69%	**92%**	88%	83%	62%	64%	70%
	>5	06%	01%	09%	03%	09%	12%	03%	03%	03%	06%	07%	09%
rdis	≤ 2	89%	75%	73%	88%	84%	82%	**91%**	88%	**84%**	71%	68%	55%
	>5	05%	08%	08%	03%	07%	08%	**02%**	05%	09%	**02%**	05%	02%

Table 5.2: Distribution of *multiplicative errors* across all proposed models in this thesis, AuPASS(AU), QHist(QH), QHist++(QH++) and QHistComp(QHC) algorithms. The system that has most number of queries with multiplicative error ≤ 2 and least number of queries ≥ 5 are considered to be the most performant. This means, that for each dataset, higher numbers on the first row is better and lower number on the last line(representing queries that had a multiplicative error more than 5) is better. The winners for the dataset for a particular type of query is marked in bold.

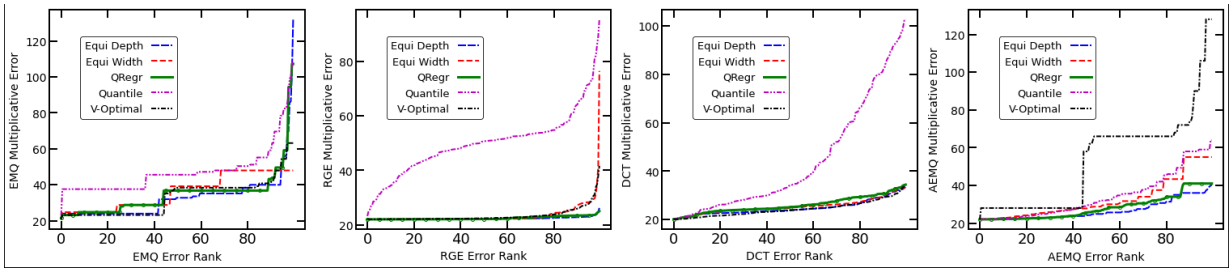
		AU			QH			QH++			QHC		
		EMQ	DCT	RGE	EMQ	DCT	RGE	EMQ	DCT	RGE	EMQ	DCT	RGE
qcew	≤ 2	67%	76%	70%	86%	**90%**	**88%**	81%	86%	84%	**91%**	**90%**	84%
	>5	02%	04%	05%	**01%**	**01%**	**01%**	06%	04%	04%	03%	04%	09%
bul	≤ 2	62%	64%	70%	89%	87%	86%	88%	**91%**	**89%**	**92%**	88%	83%
	>5	06%	07%	09%	**02%**	04%	02%	03%	**01%**	**01%**	03%	03%	03%
rdis	≤ 2	71%	68%	55%	89%	85%	83%	88%	84%	82%	**91%**	**89%**	**84%**
	>5	**02%**	05%	02%	05%	**01%**	**01%**	03%	07%	08%	**02%**	05%	09%

5.1.4 Application Scenarios of *QHist*, *QHist++*, *QHistComp* and *AuPASS*

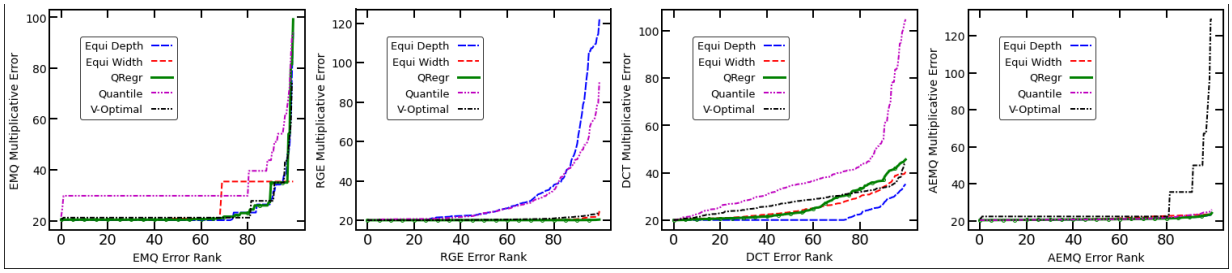
From the analysis of the QRegrArea as shown in Figure 5.2, it is clear that the overall errors are in better control with the **QHist** and **AuPASS** family of models and Figure 5.3 shows that among the QHist family, *QHistComp* performs at par in terms of histogram size. Based on the data, we will recommend the use of *QHist++* for systems with critical requirements for overall control in error and *QHistComp* for systems where the workload needs to access a lot of attributes on a frequent basis and thus needs statistics for the same

Table 5.3: Table showing percentiles of multiplicative errors across different algorithm families (qe being q-error or 100th quantile in the sorted error array), compared to that of the QHist models. These have been averaged over 100 runs across medium sized tables. For large tables, V-Optimal algorithm was turning out to be prohibitively time consuming and did not complete in certain cases. The lowest q-errors have been shown in bold and 19 out of the 24 cases that have been tested, the QHist family of histograms have performed as the best.

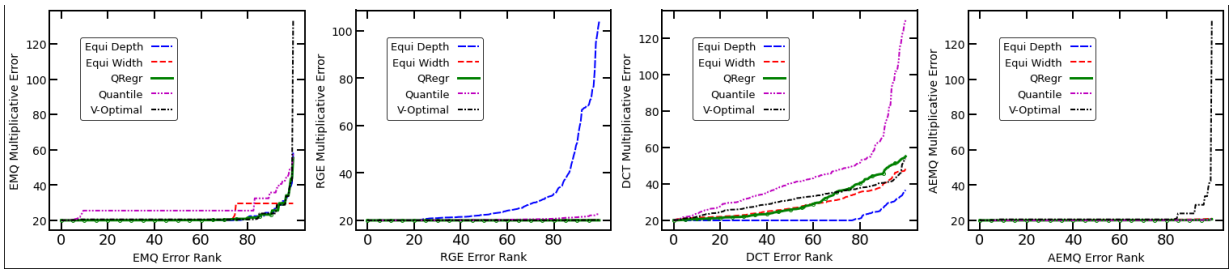
Distribution ↓		**Equi Width**				**Equi Depth**				**Avg. of V-Optimal family**				**Quantile based**				**Avg. of QHist family**			
	→ Error Percentile	**Q90**	**Q95**	**Q99**	**qe**	**Q90**	**Q95**	**Q99**	**qe**	**Q90**	**Q95**	**Q99**	**qe**	**Q90**	**Q95**	**Q99**	**qe**	**Q90**	**Q95**	**Q99**	**qe**
Normal	**EMQ**	2	3	4	5	2	3	6	7	2	2	4	3	50	50	50	51	1	2	2	**3**
	RGE	1	1	1	**1**	1	1	1	**1**	1	1	1	**1**	50	52	55	56	1	1	1	**1**
	DCT	1	1	15	28	48	58	83	90	37	42	49	56	1734	1965	2314	2619	1	2	2	**2**
	AEMQ	1	1	1	**1**	1	1	1	**1**	1	1	1	**1**	51	52	56	56	1	1	1	**1**
Uniform	**EMQ**	1	1	1	**2**	12	23	36	79	1	1	2	**2**	50	50	53	82	2	2	2	**2**
	RGE	1	1	1	35	4	4	4	**4**	10	11	11	11	53	53	53	54	1	1	1	**2**
	DCT	1	1	2	**2**	1	1	1	**1**	2	2	3	3	90	91	92	92	2	3	3	5
	AEMQ	1	1	1	**1**	2	3	6	7	2	2	3	3	50	50	50	51	2	3	5	6
Pareto	**EMQ**	1	1	1	**1**	1	1	1	**1**	1	1	1	**1**	50	52	55	56	1	1	1	**1**
	RGE	1	1	15	28	48	58	83	90	37	42	49	56	1734	1965	2314	2619	1	2	2	**2**
	DCT	1	1	1	**1**	1	1	1	**1**	1	1	1	**1**	51	52	56	56	1	1	1	**1**
	AEMQ	1	1	1	**2**	12	23	36	79	1	1	2	**2**	50	50	53	82	2	2	2	**2**
Random	**EMQ**	1	1	1	35	4	4	4	**4**	10	11	11	11	53	53	53	54	1	1	1	**2**
	RGE	1	1	2	**2**	1	1	1	**1**	2	2	3	3	90	91	92	92	2	3	3	5
	DCT	375	381	384	385	326	330	336	**337**	362	371	378	381	19260	19301	19371	19374	373	378	380	380
	AEMQ	271	289	304	**324**	271	289	304	**324**	271	289	304	**324**	13513	15069	16326	16575	271	289	304	**324**
Cauchy	**EMQ**	1397	1594	1800	1832	1368	1592	1796	1802	1296	1413	1714	1825	28502	30813	34017	35833	1441	1605	1762	**1788**
	RGE	2	2	2	**3**	2	2	2	**3**	2	3	7	7	2	2	2	**3**	2	3	3	**3**
	DCT	1	2	2	**2**	1	2	2	**2**	1	2	2	**2**	1	2	2	**2**	1	1	2	**2**
	AEMQ	9	11	13	14	1	1	1	**1**	2	2	2	3	4	4	4	5	3	3	3	3
Zipfian	**EMQ**	2	2	2	**2**	1	1	1	**1**	2	2	2	2	1	1	1	**1**	1	1	1	**1**
	RGE	2	2	2	**2**	1	2	2	**2**	2	2	2	**2**	1	2	2	**2**	1	1	2	**2**
	DCT	1	1	1	**2**	1	1	1	**1**	6	6	6	6	1	1	1	**1**	1	1	1	**1**
	AEMQ	4	8	25	48	1	1	1	**2**	3	4	8	11	5	6	7	8	13	24	70	136



(a) Average Error ranks across small tables (card $<$ 100000)



(b) Average Error ranks across medium sized tables (card $>$ 100000 and card $<$ 1000000)



(c) Average Error ranks across large tables (card $>$ 1000000)

Figure 5.1: Rank error distribution of the multiplicative errors for the AEMQ, EMQ, RGE, DCT queries over state of the art algorithms(EquiDepth, EquiWidth, Quantile, VOptimal) against average over QHist family(average of the three proposed models is bucketed as QRegr) of algorithm over the chosen open source datasets. More experimentation on different distributions have been shown later. These experiments were run across 20 different distributions and 175 workloads. From the view point of average multiplicative errors, the QHist family of algorithms have shown to be superior across different groups of runs. The only algorithm that has fared better than QHist family on the DCT queries, but then their performance on RGE queries outweigh their small improvements over QHist in DCT queries.

on a regular basis to be loaded in memory. If the most frequent queries are involved with a small set of attributes, then the recommendation would be to use *QHist++*.

For statistics on data creation, during the ingestion of the data, we are recommending **AuPASS**. Given the loss of data with sampling, the **AuPASS** algorithm cannot always

Table 5.4: Q-Regression $\langle \alpha, \beta \rangle$ pairs, shown for the various datasets using state of the art models. The next table has the values for the QRegr family of algorithms. The size distribution in terms of QRegrArea has been shown later.

Q-Regression pairs				
Dataset	**EquiWidth**	**EquiDepth**	**V-Optimal**	**Quantile**
qcew_EMQ	⟨-4.2, 3.5⟩	⟨-5.5, 4.9⟩	⟨-4, 6.6⟩	⟨-7.5, 1.9⟩
bul_EMQ	⟨-5.1, 3.4⟩	⟨-4.3, 3.5⟩	⟨-3.6, 4.7⟩	⟨-6.9, 2.7⟩
rdis_EMQ	⟨-7.3, 3.3⟩	⟨-8.5, 4.3⟩	⟨-5.2, 3.6⟩	⟨-5.5, 1.5⟩
qcew_RGE	⟨-21.1, 7.5⟩	⟨-19.5, 8.9⟩	⟨-14, 9.6⟩	⟨-6.5, 3.4⟩
bul_RGE	⟨-23.1, 7.4⟩	⟨-15.3, 7.9⟩	⟨-13.7, 8.7⟩	⟨-4.5, 2.9⟩
rdis_RGE	⟨-27.3, 9.3⟩	⟨-19.5, 8.1⟩	⟨-15.4, 10.6⟩	⟨-5.5, 0.9⟩
qcew_DCT	⟨-19.2, 7.1⟩	⟨-15.9, 8.6⟩	⟨-13, 6.6⟩	⟨-4.9, 3.1⟩
bul_DCT	⟨-15.2, 7.1⟩	⟨-13.3, 6.5⟩	⟨-14.6, 7.7⟩	⟨-7.1, 2.4⟩
rdis_DCT	⟨-17.3, 8.7.3⟩	⟨-18.5, 7.3⟩	⟨-15.3, 9.6⟩	⟨-8.5, 1.1⟩

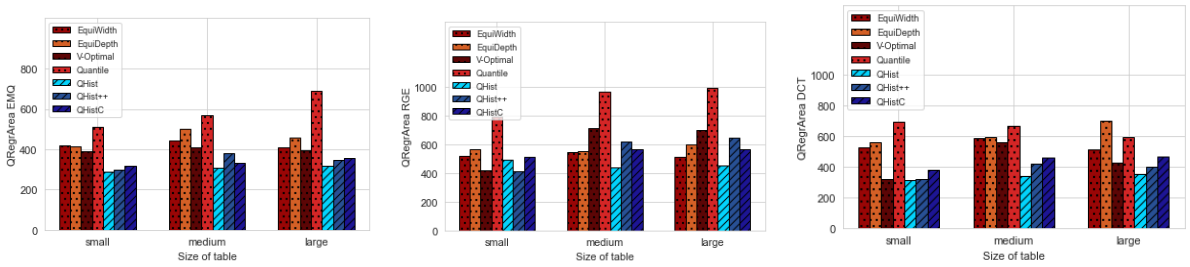
Table 5.5: Q-Regression $\langle \alpha, \beta \rangle$ pairs, shown for the various datasets using state of the art models compared to QHist, QHist++, QHistComp and AuPASS algorithms. The size distribution in terms of QRegrArea has been shown later.

Q-Regression pairs				
	QHist	**QHist++**	**QHistComp**	**AuPASS**
qcew_EMQ	⟨-1.6, 1.2⟩	⟨-2.5, 1.7⟩	⟨-1.3, 1.5⟩	⟨-1.5, 2.5⟩
bul_EMQ	⟨-2.7, 1.1⟩	⟨-2.7, 1.5⟩	⟨-3.4, 1.6⟩	⟨-3.2, 1.9⟩
rdis_EMQ	⟨-2.2, 1.2⟩	⟨-2.3, 1.9⟩	⟨-3.6, 1.1⟩	⟨-2.4, 2.6⟩
qcew_RGE	⟨-4.6, 1.3⟩	⟨-3.5, 1.7⟩	⟨-4.3, 2.3⟩	⟨-2.4=9, 1.1⟩
bul_RGE	⟨-3.7, 3.3⟩	⟨-4.7, 2.7⟩	⟨-5.3, 6.6⟩	⟨-5.4, 2.6⟩
rdis_RGE	⟨-2.6, 2.2⟩	⟨-2.5, 1.9⟩	⟨-5.6, 3.1⟩	⟨-4.4, 2.6⟩
qcew_DCT	⟨-1.6, 1.1⟩	⟨-2.1, 1.5⟩	⟨-1.2, 1.3⟩	⟨-1.3, 0.6⟩
bul_DCT	⟨-1.9, 1.0⟩	⟨-2.3, 1.3⟩	⟨-2.4, 1.3⟩	⟨-1.2, 1.6⟩
rdis_DCT	⟨-1.2, 1.1⟩	⟨-2.1, 1.8⟩	⟨-3.6, 1.1⟩	⟨-1.1, 1.1⟩

Table 5.6: The Table depicts the comparative analysis of the three proposed solutions in the paper. Here we show the average build time, in-memory footprint and average and tail results for the *q-error* metrics for *QHist*, *QHist++*, *QHistComp* and *AuPASS*

	QHist	**QHist++**	**QHistComp**	**AuPASS**
Memory(in Kb)	4.9	4.1	3.1	2.1
Avg Build Time(in ms)	989	1004	1500	322
Avg Tail q-error	11	10.1	16	17.6
Avg Q-error	7.5	7.9	8.4	10.5

guarantee errors with confidence and thus will be useful only at the load time, where all the tuples are seen at least once by the system. There could be efforts in future to be able to



(a) EMQ (b) RGE (c) DCT

Figure 5.2: QRegrArea values shows the cumulative distribution function of the multiplicative errors expected for QHist family of models to be superior to that of the state of the art models. The QHist models have all performed superior to the state of the art systems on EMQ, DCT and RGE queries. V-Optimal comes close to QHist algorithm in terms of error performance in most cases except RGE queries.

create approximate solutions with sampled data as well, but that has not been attempted by this thesis.

Table 5.6 shows combined analysis of the 3 different models under test. The results confirm our aforementioned analysis and theoretical benefits of each of the models. On

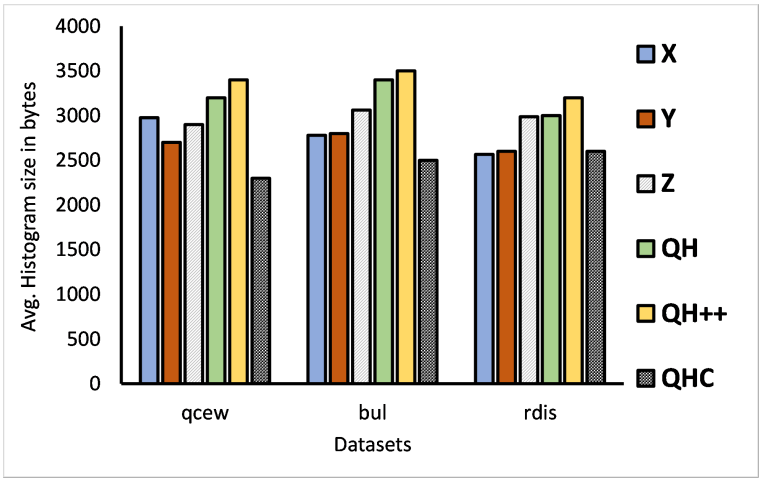


Figure 5.3: Figure showing the average size of histograms created from QHist family of algorithms and the state of the art. The performance of QHistC(QHistCompressed) is the one that is closest to the state of the art in size as the default algorithm is choosing to reduce only upto a certain number of steps. Algorithms like V-Optimal will approach it greedily and will compress in a higher ratio than the others. With QHist family, the algorithm will lose quality with forced compression.

a system with workloads that primarily use only a few attributes, working with the non-compacting histogram solutions will provide better control over the *q-error*. For workloads where a lot of ad hoc queries are expected and a larger set of attributes will contribute to the queries, the optimizer will need a large number of statistics objects in memory during the period of optimization. In such a case, with a small hit to the error, one can achieve about 23% size benefit in terms of histogram size by moving to *QHistComp*. For systems where system restarts are part of daily workload and scaling up and down as per business hours and dynamic scaling tied to workload, database restarts could lead to recomputations for in-memory data. In such cases, *QHist++* will be a solution that will provide the speedup and error guarantee comparable to *QHist* and have a lower footprint.

Chapter 6

Greener Database Management Services

6.1 Motivation

Energy Reduction is the new buzzword in the database community [22] and [21].

The energy efficiency parameters for data centers has become a talking point in the global power management circles in the recent years due to three different factors, (i) Environment (ii) High Economical footprint (iii) Performance impact.

Data Centers affect all the above three parameters due to more than one reason [66]. It is estimated that a typical data center uses as much energy as 25000 households [21]. They consume up to 300 times the energy needed by a regular office space [21]. There is a understanding amongst the industry leaders, that it takes a little below 5 years for the power requirements of data centers to double. A study by W. V. Heddeghem [31], the *Compound Annual Growth Rate(CAGR)* for this growth in United States between 2007 and 2012 was 4.4%.

Firs the first analysis of annual power consumption over a decade ago, a lot of initiatives have been launched on energy-efficiency for intensive-workload computation covering individual hardware components, system software, to applications.

A broad categorization of the power consumption can be done as follows:

1. Infrastructure usage Servers, networking infrastructure, storage etc.

2. Maintenance Systems Cooling and power conditioning systems.

An interesting point to note is that the server usage requirements can drive the cooling costs further. Most of the literature that we referred to keeps the compute usage costs between 32% and 45% of the entire power consumption, the higher side takes into account even the impact of higher network throughput due to the sub-optimal memory usage and redundancies. Outside of machine learning, training of models, almost all the workload in the wild are a form of database or dataset computation.

As detailed in the introduction section of this paper [22], this computation is mainly ensured by query optimizers by picking the most efficient plans. A bad query could take hours to compute, it could cause a IO(Input Output)storm, could deadlock and eat up the entire parallelism capabilities of the system and take up the system memory, whereas a good plan could do the same work in miliseconds.

The current versions minimize inputs-outputs operations and try to exploit RAM as much as possible, by ignoring energy. A couple of studies proposed the integration of energy into query optimizers that can be classified into hardware and software solutions. Several researchers have the idea that the operating systems and firmware manage energy and put software solutions in the second plan. This does not distinguish between tasks of operating systems and DBMSs.

Whereas, we have shown with our work that a combination of the two different strategies as described in the chapters before, when merged together can reduce this footprint.

6.2 Experimentation

In the journey to create a greener database management services, we selected two different database systems that have been used in industry for a fairly good amount of time. They are : **PostGres** and **MonetDB**.

These two database management systems have been written in **C** language and are capable of running queries in parallel. The only difference between them is the base storage models of the two access method subsystems. **PostGres** uses row based storage and **MonetDB** uses column based storage systems. The description of the two storage systems is beyond the scope of this documentation. The benefits provided by the columnar storage enables **MonetDB** to be a analytical solution while the **PosteGres** system is able to handle transactional and analytical queries, although it does not offer certain speedups of the other system.

6.2.1 PostGres Query engine

It is a row-store DBMS supporting object-relational databases [7]. The system uses the server/client model and supports the standard database languages [24]. It offers many advanced functionalities such as user-defined types, table inheritance, sophisticated locking mechanism. The support of a parallel query involves multiple background worker processes.

There is a back-end process that handles all queries issued by the connected client. This back-end consists of the following subsystems:

1. **Parser:**

   it checks the query syntax expressed in a high-level query language like SQL to determine whether it is well formulated according to the grammar rules of the query language.

2. **Analyzer:**

   The query must be validated by verifying that all attributes and relationship names are valid and semantically significant in the schema of the database.

3. **Rewriter:**

   Using transformation rules, an internal representation of the query is then created (query tree).

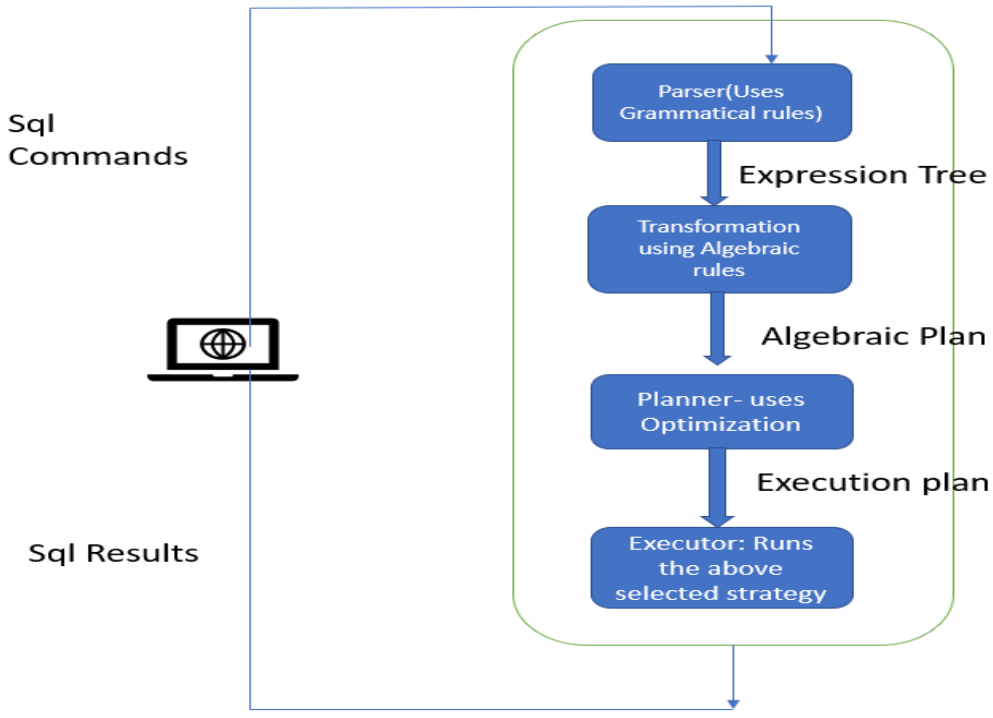


Figure 6.1: PostGres Engine internals. In planners that have a cache based plan reuse strategy, the planner can be bypassed.

4. **Planner:**

   It generates the cheapest plan tree that can be executed from the query tree.

5. **Executor:**

   A query has many possible execution strategies, and the selection of the best plan is usually conducted by cost model-driven strategies.

6.4 summarizes the different phases of the PostgreSQL query processor.

6.2.2 MonetDB

Due to the prowess of columnar processing in analytical queries, **MonetDB** was initially created to support datawarehouses alone [64]. Internally, the design, the architecture and the implementation of MonetDB reconsidered all aspects and components of classical solutions that have been in the foray for the last five decades. The **MonetDB** team has been able to piggyback on the architecture and technology by exploiting effectively the potentials of modern hardware.

Storage Model

This is where **MonetDB** takes a significant deviation of traditional database systems. It uses the "decomposed storage model" (DSM) which represents relational tables using vertical fragmentation, by storing each column in a separate #surrogate, value# table, called *binary association table (BAT)*. The left column (the surrogate or object-identifier (oid)) is called the head, whereas, the right column is the tail.

During the query evaluation, columnar storage is used for all the intermediate results. Only just before sending the final result to the client, $N - ary$ tuples are constructed.

Query execution model

The **MonetDB** kernel is an abstract machine, programmed in the *MonetDB Assembly Language (MAL)*. The core of MAL is formed by a closed low-level two-column relational algebra on BATs. N-ary relational algebra plans are translated into two column BAT algebra and compiled to MAL programs. These MAL programs are then evaluated in an operator-at-a-time manner. 6.2 shows the internal design of MonetDB. MonetDB's query processing scheme is centered around three software layers:

1. The top layer or front-end provides the user level data model and query language. The query language is first parsed into an internal representation (e.g., SQL into relational algebra), which is then optimized using domain-specific rules.

2. The middle layer or back-end consists of the MAL optimizers framework and the MAL interpreter.

3. The bottom layer or kernel provides BATs as MonetDB's bread-and-butter data structure, as well as the library of highly optimized implementations of the binary relational algebra operators.

MonetDB has a significantly different way to handle parallelism. The sequential execution plan is generated firstly and parallelization as a result of the second optimization phase. The individual MAL operators are marked as either "blocking" or "parallelizable".

Both the optimizers will alter the plan by splitting up the columns of the largest table into separate chunks, then executing the "parallelizable" operators once on each of the spliced chunk, and finally merging the results of these operators together into a single column before executing the "blocking" operators

6.3 Correlating Energy and performance

The next significant thing that we need to unfurl to the reader is our Energy calculation model. We followed our power modelling from the work done by A. Roukh et al [68] and from the work done by Dembele, Simone Piere et al [22]. The models have been

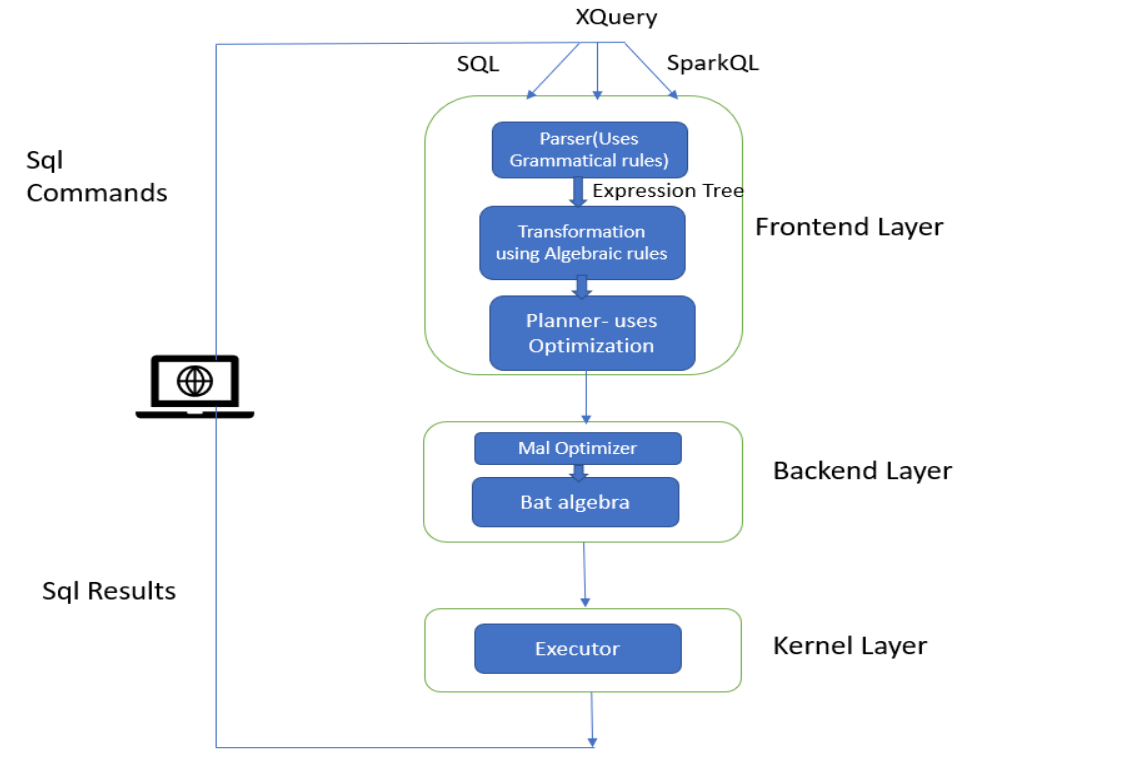


Figure 6.2: MonetDB engine is far different than the traditional database management systems of the past. As can be seen in this figure, the Kernel Layer is isolated from the algebrizer and planner layer by the intermediate MAL processing layer that it uses.

created to quantify the gains realized from the usage of the better quality statistics under the combination of **QRegression** and **AuPASS** model.

6.3.1 Preliminaries

In this section, we give the background to propose green query processors.

6.4 A. Energy Efficiency as a Non Functional Requirements

Definition 1: *Energy* (**E**) is a measurement (in **Joules**) of the ability of something to do work. A number of sources of energies are available, like kinetic energy, magnetic energy, thermal energy. It can be transformed from one type to another. In our study, we consider electrical energy. Energy is dependent on time.

Definition 2: *Power* (**P**)is defined to be the rate at which work is performed, or the derivative of work over time. Watts is used to define as the unit of *Power*. In electronics, power is defined as the amount of energy consumed per unit of time by the system. *Work (W)* is related to the amount of energy transferred in or from a system by a force. Formally, energy and power can be defined as follows:

$$P(t) = \frac{dE(t)}{dt}$$
$$E(t) = \int_0^{t_0} p(t)dt$$

where P, t, and E represent, respectively, a power, a period, and energy. Since it is hard to guarantee the accuracy of energy measure, in this paper we use the average power representing the average power consumed during the execution of the workload.

Definition 3: Energy efficiency (**EE**) expresses the optimal use of energy to offer the same service. It is expressed by [36]:

$$EE = \frac{\text{Energy utilization}}{\text{Total input energy}} = \frac{\text{Performance}}{P}$$

It is clear that there are two ways to improve the total energy requirements. Either by reducing the input energy or by increasing the performance of the system. Both can be achieved by optimizing the right way. We will summarize the demonstration of the efficiency improvement with the use of a much simpler setup.

6.4.1 Experiment setup

The datasets used are same as the ones used in the section 5.1.3. The machine used is a **Intel(R) Core(TM) i7-7700 CPU, 4 cores, 8 logical cores, 3.60GHz** processor, with **Bios Version HP N51 Ver. 01.72** and **System SKU is L8T12AV**. The machine comes with the Embedded Controller version 5.5. **4 DDR4 8 GM DIMM RAM** cards are installed. We are using a monolithic subsystem so that the network interference or the error margins introduced by network traffic is minimized.

For the power consumption portion, we used a standard smart meter, Kill A Watt, P4400, which plugs into the standard wall socket. The device is able to handle a load of 1875 **Amps** and a max voltage of 125 **Volts AC**. 6.3 shows the device when plugged in.

The Operating System on the machine was **Red Hat Linux 8(RHEL)** and the **Post-Gres** version 13 was used and **MonetDB** version 11.43 was utilized in the setup.

6.4.2 Query and Power performance

The power consumption setup have been highly discussed in the energy modelling community [66], [6], [77], [54], [13], [51], [79], [74]. In order to prove our hypothesis, we will need to quantify and create a cost model.

Since the focus is not on creating a dynamic power training module, that has a feedback cycle, that will choose the most optimal power profile for a workload, the target here is to provide concrete evidence of power savings when query optimizers have the best possible cardinality estimations.

We present the query performance at first, before showing the power consumption graphs subsequently 6.4 and 6.5. The images below show that for a given batch of *SPJ (Select*

Figure 6.3: A Kill A Watt, P4400 installed as a smart meter captures the power consumption of the monolithic system.

Project and Join Queries) on each of the two database systems used under the influence of correct statistics.

MonetDB, in terms of performance will always have the edge when it comes to analytical or read-only queries, and it works on only the attributes of a relation that are involved in the query. A lot of the performance impact on the **PostGres** end is due to the page thrashing that happens, as it has to read the entire tuple from the storage subsystem. This impact can be amplified in datacenter setups where the storage lies on a different power or fault domain in a nearby rack. Sometimes, they are not collocated due to network configurations. In those cases, the experiments could get a significant bias towards those numbers. Under the constrained setup of our experiment, the gap between the **PostGres** and **MonetDB** is relatively contained.

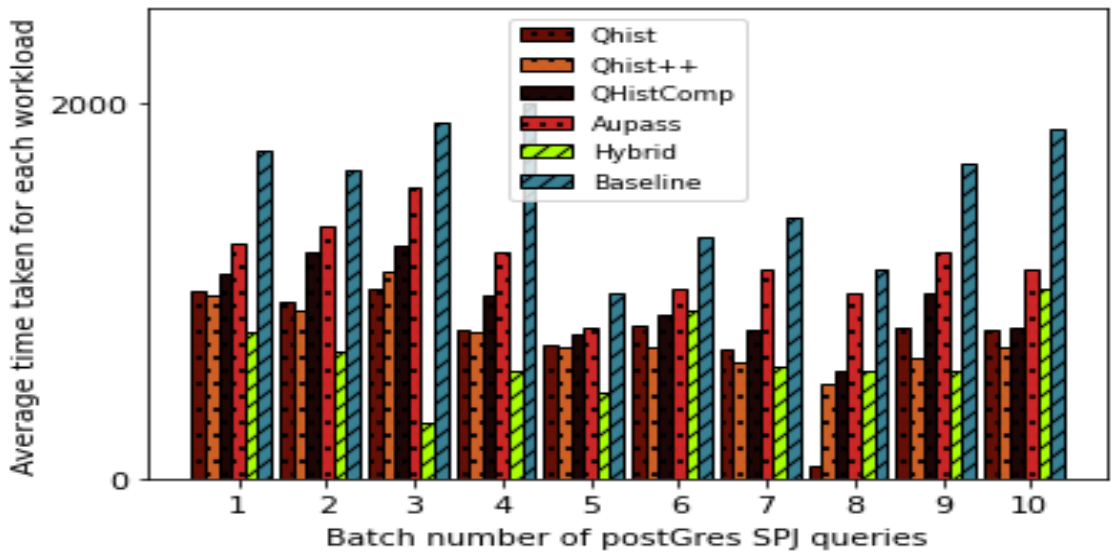


Figure 6.4: Avg performance in terms of milliseconds for a batch of workloads combining a mix of Select Project and Join queries on a PostGres System. The performance of the hybrid model is consistently faster in these query types. A smaller write workload was added so that the statistics needs modification and the AuPASS algorithm can kick in.

The figures 6.7 and 6.6 show that with a optimal query execution performance, without the impact of network interference, there is a steady 9% power consumption difference

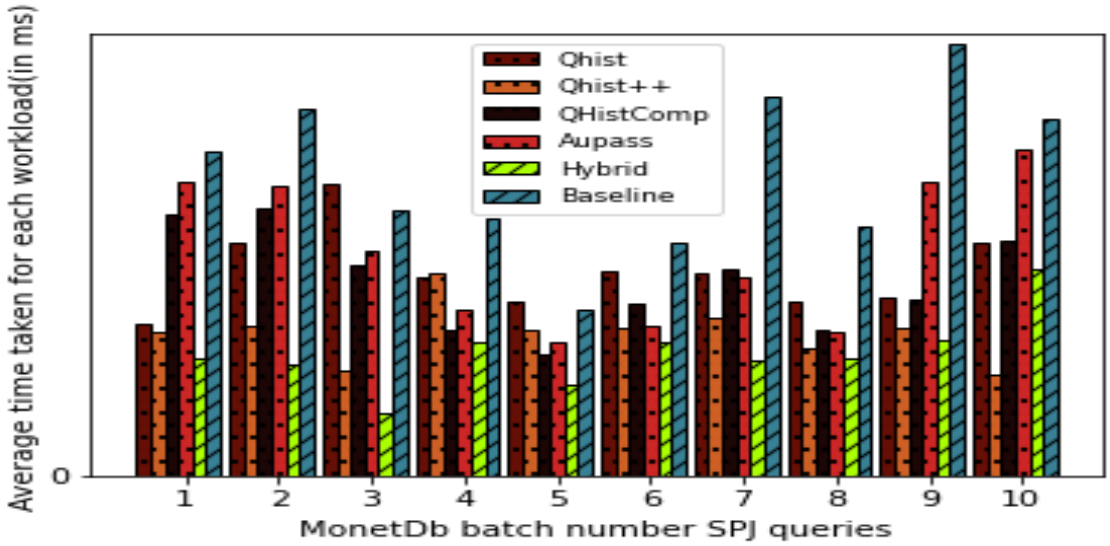


Figure 6.5: Avg performance in terms of milliseconds for a batch of workloads combining a mix of Select Project and Join queries in MonetDB. The performance of the hybrid model is consistently faster in these query types. A smaller write workload was added so that the statistics needs modification and the AuPASS algorithm can kick in.

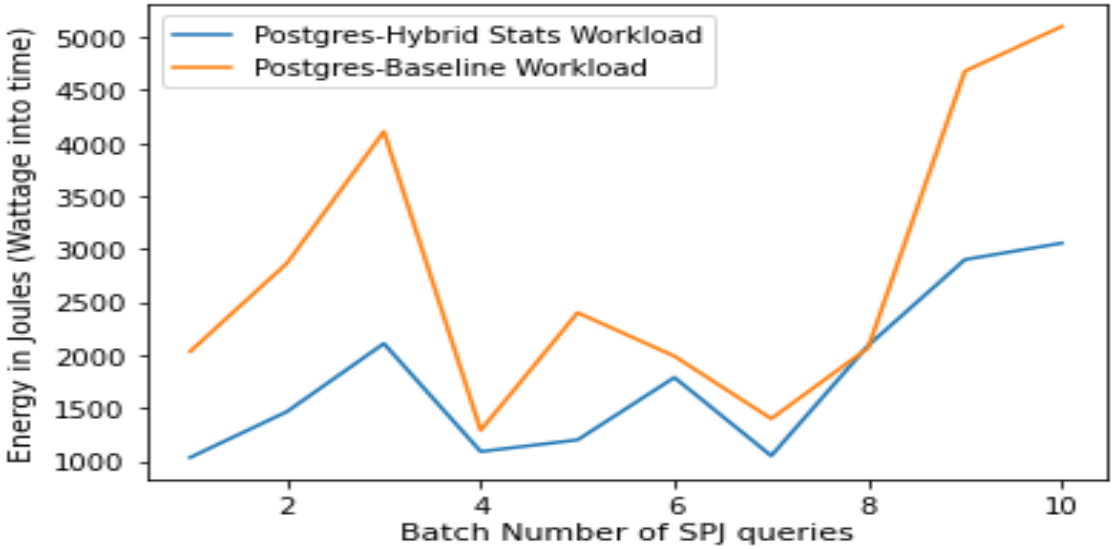


Figure 6.6: Energy usage measured as a function of time and the wattage calculated in the smart power meter against PostGres on the SPJ workloads created. This is average of the 3 datasets used.

between the workloads that are working without the impact of quality statistics versus the workloads that working with a mixture of the **QHist** family of statistics, **AuPASS** or a hybrid of the two.

Annual DataCenter power consumption is now upwards of 3% [10]. A significant portion of the workload is running data movement and analysis. Given that CPU usage consumes about 32% of the total energy consumed in a workload, this is a significant step in the right direction for all operations that need a query execution engine. The performance and energy gains from accurate statistics of course will be replaced with additional throughput as the user is always hungry for more compute, but when optimizations for energy will be seriously implemented, there will be stringency introduced. Those measures will check for redundancies at all substrates of energy consumption including what queries are absolutely needed to be run.

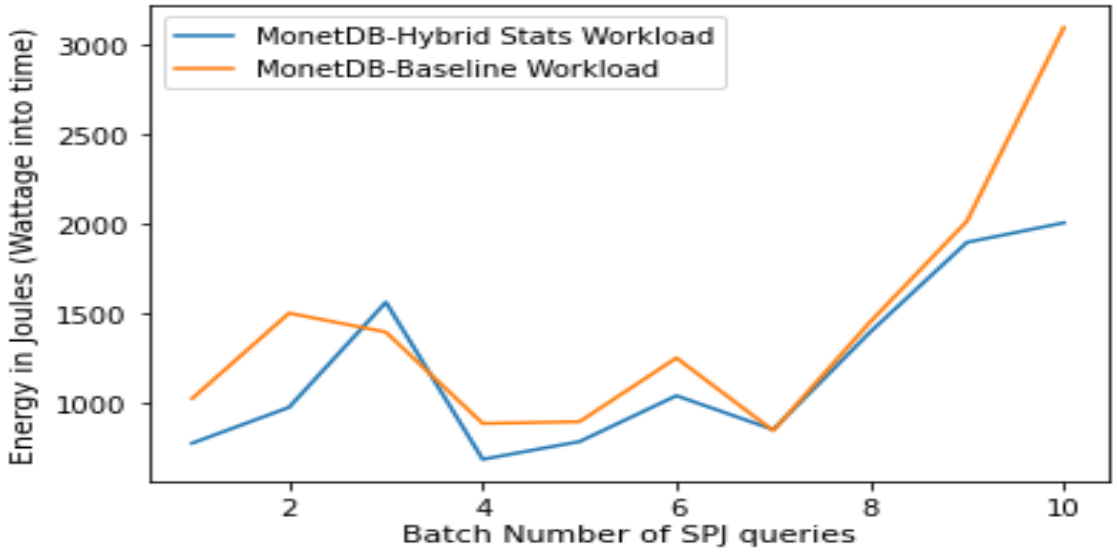


Figure 6.7: Energy usage measured as a function of time and the wattage calculated in the smart power meter against MonetDB on the SPJ workloads created. This is average of the 3 datasets used.

With the push towards a greener technology footprint, the need for each query will have to come with the power cost in the context. In those cases, the industry will be well prepared with a solution of this kind.

Chapter 7

Related Work

In this section we describe all the related work that has seen active work in regards to database statistics and its possible uses.

7.1 Related Work for Statistics

Histograms are widely used in literature and commercial implementations. Right from the first seminal paper query on optimization, by P Selinger and M. M. Astrahan, D.D. Chamberlain, R.A. Lorie and T.G. Price [69], the importance of right statistics or metadata has been discussed throughout the academia and industry.

The original papers that introduced the world to database statistics and attempts to make it better, efficient and more resilient to stringent resource requirements have been debated for decades. Here we look at work that was done for each individual sub-problem that we have tried to solve in this thesis.

7.1.1 Related work for Statistics Quality

While the quality of statistics has always revolved around *q-error* and database performance is a combination of multiple factors, that the individual parameters of statistics quality are not published by industries. The work as shown in our Energy and approximate

query processing solution has been discussed in the previous chapters, extensively. This treatment has been missing in available literature to a large extent.

Since the quality of statistics cannot help a system with poor resources in performing better in orders of magnitude, the point is moot for systems that have similar auxiliary problems. Same can be the case for a system with poor design schema, or network inefficiencies in the case of distributed systems. Another big factor is that of the case where the optimizer has inherent issues. Some optimizers use a heuristic based approach and let statistics take the back stage. In those cases, it will be difficult for a quality histogram to even create the desired effects.

Here we present all the state of the art that has existed over the years.

State of Art

Variable Range Histograms or equi-depth histograms are discussed in [38] and [48]. Serial Histograms are discussed in [36] and [35]. The main focus of the paper is to be able to create optimal histograms that allow the optimizer to focus on the most frequent elements. End-biased histograms store the high frequency and lowest frequency elements in individual buckets and thus adds to the storage requirements for the histogram. Variable-count or equi-width histograms are discussed in [38] and [48]. They are easier to maintain than equi-depth histograms. The variance within the buckets is higher in its case. Equi-depth histograms work well with range queries when the data is skewed. There are assumptions about uniform distributions of frequencies which produce estimation errors for the non-frequent terms. V-Optimal histograms are described in [37]. V-Optimal histograms are computationally intensive. There were cases where in-memory statistics needed to be generated for correlations

in case of join queries and predicates, where the computation time for a query involving multiway joins and predicates became computationally non-trivial due to the time it took to generate a V-Optimal histogram. Quantile Based histograms are discussed in [59]. [40] mainly worked on large scale distributed systems like Hadoop and thus introduced another performance indicator in terms of communication cost. Another interesting algorithm discussing the possibility of creating histograms in big data systems as a part of the movement of data was discussed in [39]. The paper discussed the idea of using a co-processor like FPGA to offload the computation work. The FPGA accelerator analyzes the tables as they are transmitted from storage to the processing unit. These provides a mechanism to create histograms on the data from the data path. Wavelet-based histograms have been discussed in [55]. The process involves a multi-resolution wavelet decomposition for building histograms on the underlying data.

The histogram stored the cumulative data distribution and optimized space usage. The paper also attempted to offer approximate solutions to user queries. Q-error was introduced in [57]. The paper [42] had done extensive analysis on improving histograms based on the *q-error*. None of above mentioned papers have approached the problem from the point of view of analyzing a linear regression on the sorted set of the errors.

7.2 Related work for Single Pass algorithms

A lot of research on this field has been attempted during the early phases of optimizer statistics. Ben Haim et al. [5] discussed this extensively in their research. Though the original intent was to achieve the ability to distribute the work for decision tree classifiers, their essence was to quickly construct histograms at the processor levels, thus compressing

the data to fixed amount of memory. A control node is then used to find the optimal splitting and partitioning function that will allow the terminal nodes to achieve linear scaling.

Similar approaches had been attempted with SLIQ, paper by Mehta et al. [56] 1996, and its successor SPRINT by Shafer et al. [71]. The biggest shortcoming with those papers were the fact that they needed pre-sorted data. If the data in the wild is not indexable or the system does not have the compute power or memory to index, then this will be a bottleneck on the compute end as these devices are working with heterogeneous data cannot dedicate resources for a singular relation or database.

Sorting was replaced with approximate representations as in BOAT by Gherke et al. [27] and SPIES by Jin and Agrawal. [41]. Both the approaches have certain restrictions on the type of data that can be used and they are better suited for attributes with smaller domains.

One other approach that was applied was by Guha et al. [29] which could be applied for a variety of attribute types, but there is the requirement for sorting the data.

7.3 Using Statistics in more ways than one!

Multiple papers have found other uses of statistics other than just optimizing a query. These innovations can all benefit from the presence of a single pass statistics as built by the **AuPASS** algorithm, while quality statistics can always be updated using **QHist** algorithms, after the ingested data has gone through a transactional phase, where the statistic have changed significantly.

7.3.1 Improvements to Other Aspects of Query Engine

In this paper by Hans-Peter Kreigel, Peter Kunath, Martin Pfeifle and Mathias Renz [49], describe another unique way where access patterns to various indexes can be improved by using good quality statistics. New Relational index structures, as for instance the **Relational Interval Tree**, the **Relational R-Tree**, or the **Linear Quadtree**, enable efficient processing of queries on top of existing object-relational database systems. Also, there exist effective and efficient models to estimate the selectivities and the I/O cost in order to guide the cost-based optimizer whether and how to include these index structures into the execution plan. This approach adds an additional data driven decision layer, that introduces performance in the storage layer of the system. For distributed systems, the impact can be non-trivial.

By design, the schemes immediately fit to common extensible optimization and indexing frameworks, and their implementations exploit the built-in statistics facilities of the various database management systems that are getting tested. In this paper, the authors were able to show that these statistics can also be used for accelerating the access methods themselves by reducing the number of generated join partners.

The different join partners are grouped together according to a cost-based grouping algorithm. Their first experiments on an **Oracle9i** database yield a speed-up of up to 1,000% for the **Relational Interval Tree**, the **Relational R-Tree** and for the **Linear Quadtree**.

Another interesting paper, developed in *Microsoft Research* using **Sql Server** as the engine, was written by Surajit Chaudhuri and Vivek Narasayya [11]. They tuned **TPC-D**

1GB database on **Microsoft SQL Server 7.0** with 13 indexes, and a workload consisting of the 17 queries defined in the benchmark. They recorded the plans for each query without any additional statistics on columns (besides statistics on indexed columns).

After that part of the experiment, they created a set of relevant statistics for the workload and re-optimized each query and recorded its plan. They observed that in all but 2 queries, the execution plans chosen with additional statistics were different, and resulted in improved execution cost. They ended up creating a automated statistics management system to better manage the statistics they need. This introduces a great new avenue for better quality statistics to cover for missing statistics, as large systems with hundreds of thousands of tables, with more than a dozen columns cannot be expected to maintain all the statistics in a memory efficient or compute efficient manner.

Using statistics for deterministic testing

A very interesting outcome of this paper was the usage of stochastic testing methodology as developed by Don Stultz [72]. Our experiments with the **TPC-D** were all done using this methodology. Deterministic testing of any database management system is human intensive and never complete the entire domain space of the data types and the nature of data. With the proliferation of modern data types, it has become a even bigger challenge in the database communities. They introduced a system called **RAGS** to build a system that was a million times faster than any human or hand generated testing infrastructure.

Automated queries can be built using this system and this has the added benefit of being repeatable and deterministic when testing two similar systems.

7.3.2 Other possible uses of optimizer statistics

A paper by Qiang Zhu et. al, [81] works on another wonderful mechanism to capture statistics by piggy backing at runtime, so that the data to be analyzed is already in-memory. They then slowly build the entire statistics using an interleaving algorithm. Since the solution proposed by us in **AuPASS**, we have already solved this problem as we will be creating the statistics when the data is ingested. A further improvement that can be worked upon using the above strategy will be to piggy back on the query runtime, when a significant amount of modifications have happened due to inserts, updates and deletions. Those will not be captured in a streaming fashion using our proposed solution can this integration could be meaningful.

Another paper here by Shouke Qin et al [63] have discussed the approximate processing of multi-granularity aggregate queries over data streams.

Self Tuning databases with statistics

Database administrators spend a lot of their time tuning the database knobs, settings and thresholds and adhoc changes to the workload can lead to unforeseen situations.

Since the algorithms can be used over the analytical streams generated from within the database runtime environment, it is also possible to generate quick summaries of the runtime metrics, and use that to tune database environment. This is a topic of intense research and there is a lot of interest in the industry regarding the same. Oracle has already published multiple papers on this, please refer to this paper by Belknap, Peter and Dageville, Benoit and Dias, Karl and Yagoub, Khaled, [4], [67].

Chapter 8

Conclusion

8.1 Conclusion and Future Work

We proposed a novel principle of *Q-Regression* and new metric called *QRegrArea* and we introduced the idea of a single pass, streaming algorithm *AuPASS*. We developed a novel approach to histogram creation that is able to normalize the affect of dense zones in the cumulative frequency distributions that cannot be easily approximated given the space constraints and complexity of computation. To our best knowledge, our work is the first of its kind that tries to address this problem by creating a regression line for the multiplicative errors and uses that for flattening the overall regression line to lower the *q-error* bound. Experiments have proven that compared to the state of the art, all our proposed methods yielded a significant reduction in terms of *q-error* for EMQ, RGE, and DCT. Also, the slopes of the trend lines for the multiplicative errors in our proposed methods were less steep than the state of the art models. Future improvements can be attempted with use of heterogeneous buckets and use *q-compression* buckets [42] in cases where we cannot fit our requirements. More future work is planned towards achieving a one pass algorithm that does not need the error values to be sorted when achieving bucket merge and to work on multi column statistics.

In the single pass algorithm , the use of *KLL* sketches [31] in collaboration with *Count-Min Sketches* has proven to handle multiple pain points associated with statistics generation amongst distributed query answering applications. They are low in memory footprint and don't need the data to be sorted, and they can be merged with other sketches with no reduction in the error guarantees.

The biggest improvement over the state of the art is the removal of the constraint of sorted tuples as inputs. The sorting process is heavy on the memory and can interfere with concurrent queries running on the system. Most distributed systems are not dedicated to one single application or work flow and their central resources are shared across multiple different solutions.

Bibliography

[1] Daniel Abadi et al. "The Seattle Report on Database Research". In: *SIGMOD Rec.* 48.4 (Feb. 2020), pp. 44–53. ISSN: 0163-5808. DOI: 10.1145/3385658.3385668. URL: https://doi.org/10.1145/3385658.3385668.

[2] M. M. Astrahan et al. "System R: Relational Approach to Database Management". In: *ACM Trans. Database Syst.* 1.2 (June 1976), pp. 97–137. ISSN: 0362-5915. DOI: 10.1145/320455.320457. URL: https://doi.org/10.1145/320455.320457.

[3] Maria Avgerinou, Paolo Bertoldi, and Luca Castellazzi. "Trends in Data Centre Energy Consumption under the European Code of Conduct for Data Centre Energy Efficiency". In: *Energies* 10.10 (2017). ISSN: 1996-1073. DOI: 10.3390/en10101470. URL: https://www.mdpi.com/1996-1073/10/10/1470.

[4] Peter Belknap et al. "Self-Tuning for SQL Performance in Oracle Database 11g". In: *2009 IEEE 25th International Conference on Data Engineering*. 2009, pp. 1694–1700. DOI: 10.1109/ICDE.2009.165.

[5] Yael Ben-Haim and Elad Tom-Tov. "A Streaming Parallel Decision Tree Algorithm". In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 849–872. ISSN: 1532-4435.

[6] Kashif Bilal et al. "Trends and challenges in cloud datacenters". In: *IEEE Cloud Computing* 1.1 (2014), pp. 10–20. DOI: 10.1109/MCC.2014.26.

[7] Arctype Blog. *Why is PostGres called a ORDBMS or Object Relational Database Management System*. Nov. 2022. URL: https://arctype.com/blog/postgres-ordbms-explainer/#:~:text=Because%5C%20of%5C%20its%5C%20support%5C%20for,to%5C%20integrate%5C%20with%5C%20a%5C%20database..

[8] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[9] Richard Brown, Carrie Webber, and Jonathan Koomey. "Status and Future Directions of the ENERGY STAR Program". In: *Energy the International Journal* 27.5 (Dec. 2001). URL: https://www.osti.gov/biblio/806113.

[10] Rajkumar Buyya, Anton Beloglazov, and Jemal H. Abawajy. "Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges". In: *CoRR* abs/1006.0308 (2010). arXiv: 1006.0308. URL: http://arxiv.org/abs/1006.0308.

[11] S. Chaudhuri and V. Narasayya. "Automating statistics management for query optimizers". In: *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. 2000, pp. 339–348. DOI: 10.1109/ICDE.2000.839433.

[12] McKinsey Company. *The future of big data*. Jan. 2022. URL: https://www.itransition.com/blog/the-future-of-big-data.

[13] Peter Corcoran and Anders Andrae. "Emerging Trends in Electricity Consumption for Consumer ICT". In: (July 2013).

[14] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". In: *J. Algorithms*. 2005.

[15] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". In: *Journal of Algorithms* 55.1 (2005), pp. 58–75. ISSN: 0196-6774. DOI: https://doi.org/10.1016/j.jalgor.2003.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0196677403001913.

[16] Economist Corp. *Labor, Capital and Data*. Nov. 2022. URL: https://www.economist.com/special-report/2010/02/27/data-data-everywhere.

[17] IBM Corp. *Performance and Capacity implications of big data*. Jan. 2022. URL: https://www.redbooks.ibm.com/redpapers/pdfs/redp5070.pdf.

[18] Itransition Corp. *The future of big data*. Jan. 2022. URL: https://www.itransition.com/blog/the-future-of-big-data.

[19] Seagate Corp. *Size of data will increase to 175 ZB*. Nov. 2022. URL: https://www.seagate.com.

[20] Alfredo Cuzzocrea. "OLAPing Big Social Data: Multidimensional Big Data Analytics over Big Social Data Repositories". In: *Proceedings of the 2020 4th International Conference on Cloud and Big Data Computing*. ICCBDC '20. Virtual, United Kingdom: Association for Computing Machinery, 2020, pp. 15–19. ISBN: 9781450375382. DOI: 10.1145/3416921.3416944. URL: https://doi.org/10.1145/3416921.3416944.

[21] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. "Data Center Energy Consumption Modeling: A Survey". In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 732–794. DOI: 10.1109/COMST.2015.2481183.

[22] Simon Pierre Dembele, Ladjel Bellatreche, and Carlos Ordonez. "Towards Green Query Processing - Auditing Power Before Deploying". In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, pp. 2492–2501. DOI: 10.1109/BigData50022.2020.9377819.

[23] Jennie Duggan et al. "The BigDAWG Polystore System". In: *SIGMOD Rec.* 44.2 (Aug. 2015), pp. 11–16. ISSN: 0163-5808. DOI: 10.1145/2814710.2814713. URL: https://doi.org/10.1145/2814710.2814713.

[24] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, 1989.

[25] Philippe Flajolet et al. "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm". In: *IN AOFA '07: PROCEEDINGS OF THE 2007 INTERNATIONAL CONFERENCE ON ANALYSIS OF ALGORITHMS*. 2007.

[26] Miguel R. Fornari, Joao Luiz D. Comba, and Cirano Iochpe. "Query Optimizer for Spatial Join Operations". In: *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*. GIS '06. Arlington, Virginia, USA: Association for Computing Machinery, 2006, pp. 219–226. ISBN: 1595935290. DOI: 10.1145/1183471.1183508. URL: https://doi.org/10.1145/1183471.1183508.

[27] Johannes Gehrke et al. "BOAT—optimistic decision tree construction". In: *SIGMOD '99*. 1999.

[28] Victor Giannakouris et al. "MuSQLE: Distributed SQL query execution over multiple engine environments". In: *2016 IEEE International Conference on Big Data (Big Data)*. 2016, pp. 452–461. DOI: 10.1109/BigData.2016.7840636.

[29] Anna C. Gilbert et al. “Fast, Small-Space Algorithms for Approximate Histogram Maintenance”. In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC ’02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 389–398. ISBN: 1581134959. DOI: 10.1145/509907.509966. URL: https://doi.org/10.1145/509907.509966.

[30] Jayant R. Haritsa. “The Picasso Database Query Optimizer Visualizer”. In: *Proc. VLDB Endow.* 3.1–2 (Sept. 2010), pp. 1517–1520. ISSN: 2150-8097. DOI: 10.14778/1920841.1921027. URL: https://doi.org/10.14778/1920841.1921027.

[31] Ward Heddeghem et al. “Trends in worldwide ICT electricity consumption from 2007 to 2012”. In: *Computer Communications* (Sept. 2014). DOI: 10.1016/j.comcom.2014.02.008.

[32] Benjamin Hilprecht et al. *DeepDB: Learn from Data, not from Queries!* 2019. arXiv: 1909.00607 [cs.DB].

[33] Toshihide Ibaraki and Tiko Kameda. “On the Optimal Nesting Order for Computing N-relational Joins”. In: *ACM Trans. Database Syst.* 9.3 (Sept. 1984), pp. 482–502. ISSN: 0362-5915. DOI: 10.1145/1270.1498. URL: http://doi.acm.org/10.1145/1270.1498.

[34] Intel. *Intel TSMC Tile Architecture Review*. Jan. 2022. URL: https://www.pcmag.com/news/an-intel-tsmc-cpu-intels-tile-architecture-to-mix-and-match-chip-tech.

[35] Yannis E Ioannidis. “Universality of serial histograms”. In: *VLDB*. Vol. 93. Citeseer. 1993, pp. 256–267.

[36] Yannis E Ioannidis and Stavros Christodoulakis. "Optimal histograms for limiting worst-case error propagation in the size of join results". In: *ACM Transactions on Database Systems (TODS)* 18.4 (1993), pp. 709–748.

[37] Yannis E Ioannidis and Viswanath Poosala. "Balancing histogram optimality and practicality for query result size estimation". In: *Acm Sigmod Record*. Vol. 24. 2. ACM. 1995, pp. 233–244.

[38] Yannis E. Ioannidis and Viswanath Poosala. "Histogram-based solutions to diverse database estimation problems". In: *IEEE Data Eng. Bull.* 18.3 (1995), pp. 10–18.

[39] Zsolt Istvan, Louis Woods, and Gustavo Alonso. "Histograms as a Side Effect of Data Movement for Big Data". In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD '14. Snowbird, Utah, USA: Association for Computing Machinery, 2014, pp. 1567–1578. ISBN: 9781450323765. DOI: 10.1145/2588555.2612174. URL: https://doi.org/10.1145/2588555.2612174.

[40] Jeffrey Jestes, Ke Yi, and Feifei Li. "Building Wavelet Histograms on Large Data in MapReduce". In: *Proc. VLDB Endow.* 5.2 (Oct. 2011), pp. 109–120. ISSN: 2150-8097. DOI: 10.14778/2078324.2078327. URL: https://doi.org/10.14778/2078324.2078327.

[41] Ruoming Jin and Gagan Agrawal. "Communication and Memory Efficient Parallel Decision Tree Construction". In: (May 2003). DOI: 10.1137/1.9781611972733.11.

[42] Carl-Christian Kanne and Guido Moerkotte. "Histograms Reloaded: The Merits of Bucket Diversity". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD '10. Indianapolis, Indiana, USA: ACM, 2010,

pp. 663–674. ISBN: 978-1-4503-0032-2. DOI: 10.1145/1807167.1807239. URL: http://doi.acm.org/10.1145/1807167.1807239.

[43] Evdokia Kassela, Ioannis Konstantinou, and Nectarios Koziris. "Towards a Multi-engine Query Optimizer for Complex SQL Queries on Big Data". In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 6095–6097. DOI: 10.1109/BigData47090.2019.9006445.

[44] Raghav Kaushik et al. "Synopses for Query Optimization: A Space-Complexity Perspective". In: *ACM Trans. Database Syst.* 30.4 (Dec. 2005), pp. 1102–1127. ISSN: 0362-5915. DOI: 10.1145/1114244.1114251. URL: https://doi.org/10.1145/1114244.1114251.

[45] Andreas Kipf et al. *Learned Cardinalities: Estimating Correlated Joins with Deep Learning*. 2018. arXiv: 1809.00677 [cs.DB].

[46] T. Kolajo, Olawande J. Daramola, and Ayodele Adebiyi. "Big data stream analysis: a systematic literature review". In: *Journal of Big Data* 6 (2019), pp. 1–30.

[47] Arnd Christian König and Gerhard Weikum. "Combining Histograms and Parametric Curve Fitting for Feedback-Driven Query Result-size Estimation". In: *Proceedings of the 25th International Conference on Very Large Data Bases*. Morgan Kaufmann, 1999, pp. 423–434.

[48] Robert Philip Kooi. "The Optimization of Queries in Relational Databases." In: (1981).

[49] H.-P. Kriegel et al. "Acceleration of relational index structures based on statistics". In: *15th International Conference on Scientific and Statistical Database Management, 2003.* 2003, pp. 258–261. DOI: 10.1109/SSDM.2003.1214992.

[50] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. "Optimization of Nonrecursive Queries". In: *Proceedings of the 12th International Conference on Very Large Data Bases*. VLDB '86. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, pp. 128–137. ISBN: 0-934613-18-4. URL: http://dl.acm.org/citation.cfm?id=645913.671481.

[51] Dennis Kronin. *Growth in Data Center Electricity*. Nov. 2018. URL: https://www.missioncriticalmagazine.com/articles/84451-growth-in-data-center-electricity-use-2005-to-2010.

[52] Viktor Leis et al. "How Good Are Query Optimizers, Really?" In: *Proc. VLDB Endow.* 9.3 (Nov. 2015), pp. 204–215. ISSN: 2150-8097. DOI: 10.14778/2850583.2850594. URL: http://dx.doi.org/10.14778/2850583.2850594.

[53] Vladimir Iosifovich Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals." In: *Soviet Physics Doklady* 10.8 (1966). Doklady Akademii Nauk SSSR, V163 No4 845-848 1965, pp. 707–710.

[54] Vimal Mathew, Ramesh K. Sitaraman, and Prashant Shenoy. "Energy-aware load balancing in content delivery networks". In: *2012 Proceedings IEEE INFOCOM*. 2012, pp. 954–962. DOI: 10.1109/INFCOM.2012.6195846.

[55] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. "Wavelet-Based Histograms for Selectivity Estimation". In: *SIGMOD Rec.* 27.2 (June 1998), pp. 448–459. ISSN: 0163-5808. DOI: 10.1145/276305.276344. URL: https://doi.org/10.1145/276305.276344.

[56] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. "SLIQ: A fast scalable classifier for data mining". In: *Advances in Database Technology — EDBT '96*. Ed. by Peter Apers, Mokrane Bouzeghoub, and Georges Gardarin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 18–32. ISBN: 978-3-540-49943-5.

[57] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. "Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors". In: *Proc. VLDB Endow.* 2.1 (Aug. 2009), pp. 982–993. ISSN: 2150-8097. DOI: 10.14778/1687627.1687738. URL: https://doi.org/10.14778/1687627.1687738.

[58] Gordon E. Moore. "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (2006), pp. 33–35. DOI: 10.1109/N-SSC.2006.4785860.

[59] Gregory Piatetsky-Shapiro and Charles Connell. "Accurate estimation of the number of tuples satisfying a condition". In: *ACM Sigmod Record* 14.2 (1984), pp. 256–276.

[60] Guillaume Pitel et al. "Count-Min Tree Sketch: Approximate counting for NLP". In: *CoRR* abs/1604.05492 (2016). arXiv: 1604.05492. URL: http://arxiv.org/abs/1604.05492.

[61] Viswanath Poosala et al. "Improved Histograms for Selectivity Estimation of Range Predicates". In: *SIGMOD Rec.* 25.2 (June 1996), pp. 294–305. ISSN: 0163-5808. DOI: 10.1145/235968.233342. URL: http://doi.acm.org/10.1145/235968.233342.

[62] W. H. Press et al. *Numerical Recipes in C: The Art of Scientific Computing*. second. New York, NY: Cambridge University Press, 1992.

[63] Shouke Qin, Weining Qian, and Aoying Zhou. "Approximately Processing Multi-granularity Aggregate Queries over Data Streams". In: *22nd International Conference on Data Engineering (ICDE'06)*. 2006, pp. 67–67. DOI: 10.1109/ICDE.2006.22.

[64] Mark Raasveldt. "MonetDBLite: An Embedded Analytical Database". In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. Houston, TX, USA: Association for Computing Machinery, 2018, pp. 1837–1838. ISBN: 9781450347037. DOI: 10.1145/3183713.3183722. URL: https://doi.org/10.1145/3183713.3183722.

[65] Naveen Reddy and Jayant R. Haritsa. "Analyzing Plan Diagrams of Database Query Optimizers". In: *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, pp. 1228–1239. ISBN: 1-59593-154-6. URL: http://dl.acm.org/citation.cfm?id=1083592.1083735.

[66] Suzanne Rivoire et al. "Models and Metrics to Enable Energy-Efficiency Optimizations". In: *Computer* 40.12 (2007), pp. 39–48. DOI: 10.1109/MC.2007.436.

[67] S.F. Rodd and U.P. Kulkarni. "Adaptive Self-Tuning Techniques for Performance Tuning of Database Systems: A Fuzzy-Based Approach". In: *2013 2nd International Conference on Advanced Computing, Networking and Security*. 2013, pp. 124–129. DOI: 10.1109/ADCONS.2013.49.

[68] Amine Roukh et al. "Eco-Physic: Eco-Physical design initiative for very large databases". In: *Information Systems* 68 (2017). Special issue on DOLAP 2015: Evolving data warehousing and OLAP cubes to big data analytics, pp. 44–63. ISSN: 0306-4379. DOI: https:

//doi.org/10.1016/j.is.2017.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0306437916305129.

[69] P. Griffiths Selinger et al. "Access Path Selection in a Relational Database Management System". In: *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*. SIGMOD '79. Boston, Massachusetts: Association for Computing Machinery, 1979, pp. 23–34. ISBN: 089791001X. DOI: 10.1145/582095.582099. URL: https://doi.org/10.1145/582095.582099.

[70] Raghav Sethi et al. "Presto: SQL on Everything". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2019, pp. 1802–1813. DOI: 10.1109/ICDE.2019.00196.

[71] John C. Shafer, Rakesh Agrawal, and Manish Mehta. "SPRINT: A Scalable Parallel Classifier for Data Mining". In: *Proceedings of the 22th International Conference on Very Large Data Bases*. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 544–555. ISBN: 1558603824.

[72] Donald R. Slutz. "Massive Stochastic Testing of SQL". In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 618–622. ISBN: 1558605665.

[73] Ji Sun and Guoliang Li. *An End-to-End Learning-based Cost Estimator*. 2019. arXiv: 1906.02560 [cs.DB].

[74] Ward Van Heddeghem et al. "Trends in worldwide ICT electricity consumption from 2007 to 2012". In: *Computer Communications* 50 (2014). Green Networking, pp. 64–

76. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2014.02.008. URL: https://www.sciencedirect.com/science/article/pii/S0140366414000619.

[75] Ramanathan Venkatraman and Sitalakshmi Venkatraman. "Big Data Infrastructure, Data Visualisation and Challenges". In: *Proceedings of the 3rd International Conference on Big Data and Internet of Things*. BDIOT 2019. Melbourn, VIC, Australia: Association for Computing Machinery, 2019, pp. 13–17. ISBN: 9781450372466. DOI: 10.1145/3361758.3361768. URL: https://doi.org/10.1145/3361758.3361768.

[76] The Next Web. *Python as a inefficient language*. Jan. 2022. URL: https://thenextweb.com/news/python-progamming-language-energy-analysis.

[77] Beth Whitehead et al. "Assessing the environmental impact of data centres part 1: Background, energy use and metrics". In: *Building and Environment* 82 (2014), pp. 151–159. ISSN: 0360-1323. DOI: https://doi.org/10.1016/j.buildenv.2014.08.021. URL: https://www.sciencedirect.com/science/article/pii/S036013231400273X.

[78] Zongheng Yang et al. "Deep unsupervised cardinality estimation". In: *Proceedings of the VLDB Endowment* 13.3 (Nov. 2019), pp. 279–292. ISSN: 2150-8097. DOI: 10.14778/3368289.3368294. URL: http://dx.doi.org/10.14778/3368289.3368294.

[79] Sungkap Yeo et al. "ATAC: Ambient Temperature-Aware Capping for Power Efficient Datacenters." In: *SoCC*. Ed. by Ed Lazowska et al. ACM, 2014, 17:1–17:14. ISBN: 978-1-4503-3252-1. URL: http://dblp.uni-trier.de/db/conf/cloud/socc2014.html#YeoHHL14.

[80] Fuheng Zhao et al. “KLL± Approximate Quantile Sketches over Dynamic Datasets”. In: *Proc. VLDB Endow.* 14.7 (Apr. 2021), pp. 1215–1227. ISSN: 2150-8097. DOI: 10.14778/3450980.3450990. URL: https://doi.org/10.14778/3450980.3450990.

[81] Qiang Zhu et al. “Piggyback Statistics Collection for Query Optimization: Towards a Self-Maintaining Database Management System”. In: *The Computer Journal* 47.2 (2004), pp. 221–244. DOI: 10.1093/comjnl/47.2.221.