Performance tuning Apache Drill on Hadoop Clusters with Genetic Algorithms

Improving industry standards for advanced analytics and business intelligence

Roger Bløtekjær

Thesis submitted for the degree of
Master of science in Informatikk: språkteknologi
30 credits

Institutt for informatikk
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2018

Performance tuning Apache Drill on Hadoop Clusters with Genetic Algorithms

Improving industry standards for advanced analytics and business intelligence

Roger Bløtekjær

Performance tuning Apache Drill on Hadoop Clusters with Genetic Algorithms

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

0.1 Abstract

0.1.1 Research question

How can we make a self optimizing distributed Apache Drill cluster, for high performance data readings across several file formats and database architectures?

0.1.2 Overview

Apache has introduced a new building block that's best suited on top of the Hadoop Stack called Apache Drill. Drill enables the user to perform schema-free querying of distributed data, and ANSI SQL programming on top of NoSQL datatypes like JSON or XML. As it is with the core Hadoop stack, Drill is also highly customizable, with plenty of performance tuning parameters to ensure optimal efficiency. Tweaking these parameters however, requires deep domain knowledge and technical insight, and even then the optimal configuration may not be evident. Businesses will want to use Apache Drill in a Hadoop cluster, without the hassle of configuring it, for the most cost-effective implementation. This can be done by solving the following problems:

- How to apply genetic algorithms in the most cost-effective way to automatically tune a distributed Apache Drill configuration, regardless of cluster environment.
- How do we benchmark the cluster against default Drill settings, as well as known SQL performance tests, to ensure that the algorithm is adding value to the cluster performance, measured as execution time.

0.1.3 Details

Self optimizing

The goal here is that no human intervention is needed to fine tune the Drill configuration. Once the framework / algorithm is executed on the Drill environment, it should only end execution when performance is proven to be optimized.

Genetic Algorithms

The methodology for self optimization in this thesis is genetic algorithms. A branch of evolutionary algorithms that rely on randomly distributed properties given to a population of candidates. The properties are hereditary and define the next generation of candidates, eventually resulting in a "proven optimal candidate", to solve a given problem.

Distributed file systems / cluster environments

Apache Drill can easily be run on any file system, as they are mounted when needed. As such a single node environment is entirely possible to set up, with the local file system mounted as a "Drill Bit". However this thesis focuses on current and future industry standards, and how to optimize Drill in a relevant

a big data environment. This is more representative of its intended use, and gives the thesis more value as a research field advancement.

ANSI SQL

There are similar platforms to Apache Drill, like the popular Apache Spark. Spark also provides SQL querying on data, however this is only a subset of SQL. Only Drill has the full suit of SQL programming capabilities as defines by ANSI (American National Standards Institute), which gives more in depth querying.

NoSQL

Previously NoSQL databases where known as "Non-SQL" databases, but this has now changed to be "Not Only SQL" databases as they have begun supporting SQL-like querying. The main point about them is that they are non-relational, distributed, open-source and horizontally scalable [27].

0.2 Foreword

Can be omitted if I want, keeping it for now.

Contents

Chapter 1

Preface

1.1 List of figures

1.2 List of tables

1.3 Abbreviations

TTE	Time to execute
OOP	Object Oriented Programming
HDFS	Hadoop Distributed File System
PBIL	Population Based Incremental Learning
GA	Genetic Algorithms
ER	Entity Relation
SMB	Small / Medium Business
SQL EE	Structured Query Language Execution Engine
MPP	Massively Parallel Processing
RPC	Remote Procedure Call
DSN	Data Source Name
ODBC	Open Database Connectivity
JDBC	Java Database Connectivity

Chapter 2

Introduction

2.1 Target group

This thesis covers the deep technical aspects of big data analysis and genetic algorithms - However, all techniques used will be explained in detail. Computer science students looking into infrastructure will probably feel the most at home, in terms of discussed concepts and technical terms. For the layman, there is also a word list supplied with brief explanations for technical terms used in this thesis.

2.2 Area of research

Hadoop and MapReduce are already well established technologies employed in countless applications around the world, Apache Drill however is a fairly new concept with little to no research from the community. We propose a new method of implementing Hadoop clusters with Apache Drill, automatically optimizing the performance tuning in a lightweight and low effort way. As such this is a technical thesis regarding performance tuning, and the implementation aspects of an Apache Drill cluster.

2.3 Research question - in full

2.3.1 Goal, justification and challenges

The goal for this thesis is to optimize a SQL Engine that sits on top of a distributed storage, for more cost-effective big data handling, business intelligence and advanced analytics. Businesses that handle big data will most likely perform plenty of queries every day, or even every hour. If we were to add up the amount of time spent waiting for a query to complete, for a year, and converted the hours into money - we would find it costs a lot of money to convert raw data into information. That's why saving even 10% in execution time can mean a big difference when it comes to overhead costs. So naturally there is a massive amount of research for improving execution time and other metrics, even more so when it comes to shared computing like cloud services where you pay by the minute. The SQL Engine chosen here is Apache Drill, a fairly new framework

inspired by Google Dremel, and currently the only schema-free engine to run on top of HDFS, allowing users to drill into complex data like JSON. Looking to for example Yelp, we can see that their entire database is available as JSON [32], allowing for far more complex and flexible handling of metrics than conventional ER databases. As with the other data handling engines on distributed storage platforms, there are many parameters to consider if one wants to increase performance - in fact so many that it is considered infeasible without in depth expertise of domain and engine. Considerations are:

- Type of storage (distributed or single node)
- Amount of computing nodes (cluster or not)
- Size of nodes (memory and CPU)
- Type of data (complex or not)
- Size of data (few large files or plenty small ones)
- Distribution of data (heavy skew or uniform distribution)
- Execution engine specific parameters (for which there are many)

All of these parameters work in tandem to produce a measurable performance. That's why genetic algorithms was chosen as a tool for self optimization, to take all of these parameters into consideration automatically - and simply test solutions until converging into a good one (considered optimal). Some of the parameters are statically set in this thesis, such as cluster size, to simulate a SMB environment, and to have some testing grounds. There would be no alterations needed to apply the methods presented in this thesis in either an enterprise with 1000 clustered nodes, or a single laptop, but for the sake of argument we mimic a real world use case.

2.3.2 Resarch question

How can we make a self optimizing distributed Apache Drill cluster, for high performance data readings across several file formats and database architectures?

2.4 Personal motivation

Subject

The subject for this master thesis is a natural continuation of our previous work *Hadoop MapReduce Scheduling Paradigms*, published in 2017, in the 2nd IEEE International Conference on Cloud Computing and Big Data Analysis (ISSS-BDA 2017). Back then the topic was haphazardly picked from a list of eligible ones, but the more we read into it - the more we understood the incredible use cases for Hadoop within the massive industries that are driving forces for our technological advancements. As is common in IT, levels of abstraction get added on top of proven technologies both to make implementation better, and often to increase performance. Since then Apache Drill has seen plenty of stable builds and proven itself to be potentially industry-changing in the way we handle our data - entering a schema-on-the-fly paradigm.

Apache Drill as a platform

As mentioned previously there is little to no research done on Apache Drill as a platform. Some of the reasoning might be the that some still considers it to be in the early development phase, but as of this writing there has already been one major release, **1.0** in 2015, and 12 subsequent minor releases up till **1.12** [18]. So we would argue the project is past the early development phase, and moving into the early adopter phase. But why choose to use this platform over more established ones, and what other options are there? This is discussed further in a following section: Why is this worthwhile?

2.5 Research method in brief

Throughout this thesis we will develop an entire suite of tools centered around a genetic algorithm for automatically optimizing Apache Drill on top of a Hadoop cluster, tested on big and diverse sample sets. As the framework is expected to change, all of the parameters tweaked will be in relation to the current version being used, **1.12**. The main goal is to achieve a performance increase, measured as TTE (Time to Execute) for single and/or concurrent queries. Examining previous studies on SQL performance will provide a good basis for best practices within SQL query execution optimization, and how to measure it.

2.6 Most relevant previous findings

There is little to no research done on the impact of tuning Apache Drill. However, tuning of parameterized frameworks have been done a lot in the past on industry standards like SQL, MySQL, traditional Hadoop clusters (mapreduce scheduling algorithms), and Web applications. Since Apache Drill has its own SQL execution engine, the tuning of SQL systems in previous works has value for how we will benchmark our Drill perfomance.

2.6.1 Starfish

Herodotos et al made on of the most cited papers in the Hadoop research field, when they proposed Starfish [5]. Starfish is a self-tuning system for Hadoop clusters, citing the lackluster performance of default cluster parameters to be the motivation. They designed a modular system where the main parts include:

- A profiler that analyzes jobs and determines cost estimation and the data flow.
- A novel approach to predictive tuning they named the "what-if-engine". It's job is to predict how different parameter configuration tweaks will change the performance of the system.
- A cost-based optimizer, that performs the pure tuning aspects, based on estimations from the what-if-engine.

Our project has very similar goals and motivations compared to this project, except on a framework on top of Hadoop, instead of directly on it.

2.7 Why is this worthwhile, why Drill?

2.7.1 Ease of use

It is safe to say that information technology as a subject is only getting more popular by demand. According to projections made by renown recruitment company Modis, tech employment will see a 12% growth by 2024, vs 6,5% in all other industries[13]. This means that a lot of new engineers will enter the workforce, and start building and maintaining enterprise applications. Apache Drill serves a very low barrier of entry for newcomers with its' SQL ANSI queries when handling big amounts of data, as opposed to i.e the popular Java Enterprise framework Persistence. Using Apache Drill will therefore require less training, thus increasing productivity and agility in young teams.

2.7.2 Increasing relevance of Big Data infrastructures

"From 2005 to 2020, the digital universe is expected to grow dramatically by a factor of 300, from 130 exabytes to 40 trillion gigabytes, i.e more than 5,200 gigabytes per person in 2020. Moreover, the digital universe is expected to double every two years" [22].

Real time data usage

Successful digitalization of businesses often require applications to utilize data in real time, serving customers relevant data instantaneously. *"Now, most applications are generating real-time data, so superfast data capturing and analysis within a fraction of a second are mandatory"* [24].
An interesting example of this is chat applications (instant messaging). Users of messaging apps today expect it to store data indefinitely, and deliver instant communication between users, creating an absolutely enormous global data flow. Bettercloud.com performed a study and found that [20]:

- The majority of organizations (57%) use two or more real-time messaging applications.
- 80% of Skype for Business users, 84% of Google Hangouts users, and 95% of Slack users say communication has improved because of real-time messaging.
- 56% of respondents believe that real-time messaging will displace email as their organization's primary workplace communication and collaboration tool.

From these results we see that adoption of applications using real time data is only increasing. This will provide a future need for fast and scalable infrastructure to cope with it.

Combining real-time and stored data

In reality a lot of current applications that create value combine the usage of analytics on stored data and real-time sensor data. Predictive maintenance is an example of a discipline where you analyze historical data, finding thresholds

and patterns for when components in a system break down, and then combining this knowledge with real time sensors to service the components before they fail. This saves money on planned maintenance where healthy parts get serviced, and downtime on systems where they have to be set aside for the service. In a Harvard Business Review on a company called **Servicemax** that has specialized in services like this they wrote that: *"(...) customers, on average, increase productivity by 31%, service revenue by 14% and customer satisfaction by 16%"* [23]. All of this is empowered by big data, and the utilization of information.

2.7.3 Application needs

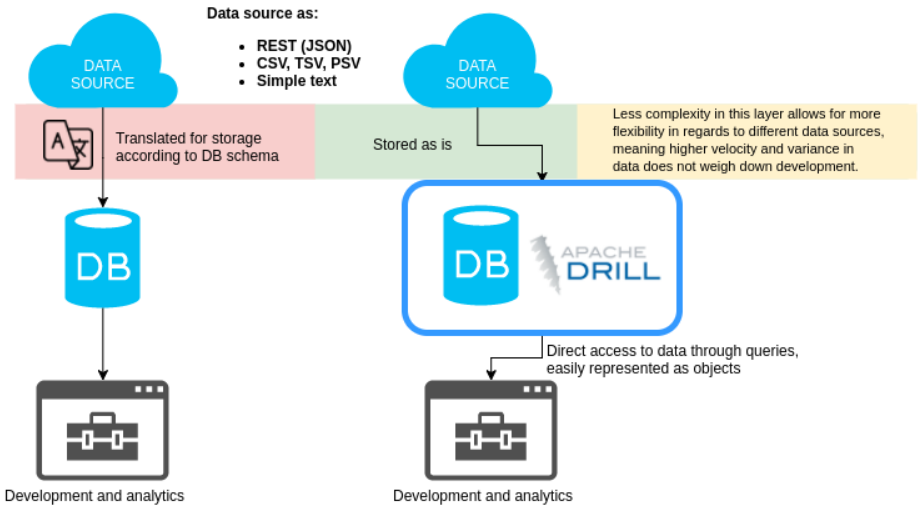


Figure 2.1: Advantages of schema-free querying

"Object and relational technologies are grounded in different paradigms. Each technology mandates that those who use it take a particular view of a universe of discourse. Incompatibilities between these views manifest as problems of an object-relational impedance mismatch" [19].
Traditional databases contain tables (entities) with relations between them. Sometimes these systems consist of thousands of tables, even going as far as tens of thousand (like this database of the human genome with 20000 tables [21]). Object-oriented programming classes is difficult and oftentimes inefficient to map directly to the raw data. Especially when there are large amounts of data, it can take a while just initializing the objects, and then again when serializing them to store them. Adopting complex storage file types like JSON for raw data would overcome this mismatch, and allow for more agile, and reactive development of enterprise applications utilizing huge data sources. The global IT industry is increasingly transitioning to deliver RESTful APIs to their customers. With Apache Drill, it is extremely easy to perform queries on the results, even joining them against several other data sources or combining results from several APIs at the same time, in real time. For application developers this will eliminate the middleware of a relational database, having to translate the data for different purposes. Instead of having one format for storing, one

for interfacing and one for application logic, Apache Drill and the schema-free paradigm will provide one single interface for all dimensions of data.

2.8 How far will this advance the field?

Advances in research

The ambition for this project is to provide a fully functional, open source light weight framework allowing companies to easily deploy Hadoop Clusters with Apache Drill without worrying about tailoring the solution or suboptimal performance. In terms of research, this is the first thesis written about performance tuning of Apache Drill, and as such will lay a foundation for the future of this field.

Advances in industry

As the previous section highlighted (Increasing relevance of Big Data infrastructures), we predict an increase in future global data collection, going as far as doubling the amount of data per person, per year. This data is only collected to add value to a business, and the revenue increase for businesses that embrace big data analytics will not go unnoticed by the industry, leading to wider adoption. Some businesses already make huge profits simply by gathering and selling information, like Google and Facebook, increasing their yearly revenue since 2011 by 289% and 1095% respectively [26]. Data is becoming the world's new natural resource [25]. To be able to more fluidly handle this data in applications, we believe agile teams will want flexible frameworks that empower more cost effective development and big data utilization. When the industry realizes Apache Drill can deliver that, this will lead to a paradigm shift to schema-free querying of data, and on-the-fly data reads without tailoring.

2.9 Structure of the report

Add comments about every chapter here.

Chapter 3

Background

"Apache Drill is one of the fastest growing open source projects, with the community making rapid progress with monthly releases. The key difference is Drill's agility and flexibility. Along with meeting the table stakes for SQL-on-Hadoop, which is to achieve low latency performance at scale, Drill allows users to analyze the data without any ETL or up-front schema definitions. The data can be in any file format such as text, JSON, or Parquet. Data can have simple types such as strings, integers, dates, or more complex multi-structured data, such as nested maps and arrays. Data can exist in any file system, local or distributed, such as HDFS or S3. Drill, has a "no schema" approach, which enables you to get value from your data in just a few minutes" [16].

3.1 History

3.1.1 Creation and adoption of Hadoop

Google created the Google File System in 2003 [10], predicting the paradigm of distributed storage. Then, the MapReduce concept for sorting / counting data was added in 2004 [11]. Hadoop was introduced as a platform in 2006 [8], and together with MapReduce, it made an impact in the industry, attracting talent and big companies eager to contribute and deploy. Yahoo was a big early adopter, being one of the first companies deploying big clusters as early as 2006 [8]. Then in 2008, Hadoop got the world record for fastest system to sort a terabyte of data [8]. After this, development accelerated with more contributers and interest was at an all time high. Since that point in time Apache Hadoop has become the most widely used platform for Big Data handling, empowering advanced analytics and business intelligence across several industries. The Hadoop stack is now highly scalable, ensures high availability of data, and utilizes parallel processing to deliver high performance data readings.

3.1.2 Google branching out with Dremel

In 2010 Google did further research on distributed data processing and published their paper on Dremel [9]. Dremel is the framework that empowers Google's current platform Google BigQuery, delivered as Infrastructure as a Service (IaaS). Among customers of BigQuery is well recognized brands like Spotiy, Coca Cola,

Philips, HTC and Niantic [12]. The success of Google BigQuery (by extension of Dremel), inspired Apache to create Drill, the framework being examined in this thesis.

3.2 Technology

3.2.1 Hadoop stack

Hadoop common

Hadoop common consists of a few select core libraries, that drives the services and basic processes of Hadoop, such as abstraction of the underlying operating system and file system. It also contains documentation and Java implementation specifications.

HDFS

HDFS (Hadoop Distributed File System) is a highly fault tolerant, distributed file system designed to run efficiently, even on low-cost hardware. HDFS is tailor made to express large files and huge amounts of data, with high throughput access to application data and high scalability across an arbitrary amount of nodes.

YARN

YARN (Yet Another Resource Negotiator) is actually a set of daemons that run in the cluster to handle how the jobs get distributed.

- There is a NM (NodeManager) that represents each node in the cluster, monitoring and reporting to the RM whether or not they are idle, and resource usage (CPU, memory, disk, network). A node in a Hadoop cluster divides its resources into abstract Containers, and reports on how many containers there are available for the RM to assign jobs to.
- There is an AM (ApplicationMaster) per job, or per chain of jobs, representing the task at hand. This AM gets inserted into containers on nodes, when a job is running.
- Finally there is a global RM (ResourceManager), which is the ultimate authority on how to distribute loads and arbitrate resources. The RM consists of two entities, the Scheduler and the ApplicationsManager.
  - The ApplicationsManager is responsible for accepting job submissions (at this point represented as an ApplicationMaster), i.e by doing code verification on submitted jobs, changing their status from "submitted" to "accepted". Once a job is accepted, the ApplicationsManager sends the ApplicationMaster to the scheduler to negotiate and supply containers for the job on the nodes in the cluster. The ApplicationsManager also has services to restart failed ApplicationMasters in containers, and retry entire jobs.

– The Scheduler is a pure scheduler in the sense that it only cares about delivering tasks to idle slots, based on free resources on the node. It calculates distributions across multiple nodes / containers, and can prioritize - i.e compute smaller jobs in front of large ones to more effectively complete job queue, even though the large job was submitted first. The Scheduler does not care for monitoring task completions or failures, simply distributing loads on the cluster.

The ResourceManager and the NodeManager form the data-computation framework. The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks [4].

MapReduce

MapReduce is the heart of a traditional Hadoop cluster, for which every other component is built around, to maximize its efficiency. It is a model for distributed processing of big data, to process and consolidate. It is defined by two stages - the mapping phase, and the reduce phase. Both of these phases require resources from the node it is run on, defined by a set amount of map slots and reduce slots. The amount of slots on a node for each task is parameterized and can be set by an administrator / or typically algorithmically. The map phase consists of parallel map tasks, that map a key to a value. All of these map tasks then consolidate into the reduce phase, where they are combined so that one key exists for all accumulated values. So for instance if we're counting cards in several decks across several nodes, they would each map something like "(Queen of Hearts, 1)". In the end the reduce task consolidates all these single queens, so it ends up looking like "(Queen of Hearts", 27)". This is a far more effective approach than for instance looping through all the decks and incrementing a key by one for each matching key we find. Especially when the data that typically gets handled by a Hadoop cluster is diverse and hard to predict.

3.2.2 Zookeeper

Zookeeper is not a part of the Hadoop core utilities, but is is often to be found in Hadoop clusters. It is used as a distributed storage platform for configuration information, synchronization and naming. While YARN handles the distribution of tasks between the nodes in the cluster, Zookeper takes care of failover, race conditions and sequential consistency. This tool looks more like orchestration frameworks such as Puppet or Chef, in that it organizes the amount of YARN nodes and distributes configurations, availability and atomicity.

Zookeeper Quorum

Even though Zookeeper is made as a distributed configuration management and failover safeguard, it can be installed on a single node. Running Zookeeper in this way is called a Standalone Operation, and give a user a interface to remotely connect to, to get some data on running services etc. However, we need Zookeper distributed across all our nodes. This is called a Quorum. In a Quorum, every node will check the health of the others. If there is a consensus among nodes

that one is down, Zookeeper is able to communicate this to YARN, letting it distribute loads across the remaining healthy nodes.

3.2.3 Apache Drill

In this thesis there won't be any in-depth explanations of the inner workings or architecture of Drill, but we'll gloss over the broader picture in this section.

What is it

Apache drill is the newest technology to be introduced in this chapter. All the previously mentioned frameworks are tried and tested - proven over time. Apache drill rests on top of all these technologies, and provides a way to query almost any non-relational database. This means that we can set up a cluster on a data lake with a very diverse data type, and still perform standard ANSI SQL queries on it. Even in formats like JSON, a simple SQL query can provide all the insights one might need. Apache Drill is inspired by Google Dremel, which as mentioned is the driving force behind their advanced Big Data Analytics tool - Google BigQuery. Apache Drill is a SQL EE for MPP, that works in almost any environment and on almost any file type. Whether querying complex data, text files or entire directories of log files, Apache Drill allows for standard SQL interfaces for reading the data.

How does it do it

The drillbit In its core Apache Drill simply consists of a Daemon service called a Drillbit. The drillbit can be installed on any server or client device that has data, or on any number of nodes within a Hadoop cluster, to perform SQL queries. This means that Drill is an entirely independent piece of software, able to perform its intended task without any environment specific needs. Thus Drill does not use YARN, and simply uses hadoop for the distributed storage. To run MPP in a cluster however, Drill is dependent on Zookeeper.

Querying drill In addition to its flexible installation needs, the query inputs to Drill can come from a wide variety of sources - JDBC, ODBC, a REST interface, and from a C++ or Java API.

Key components

- An RPC end point to receive queries and communicate with clients
- An SQL parser that interprets the SQL input and outputs a logical plan
- An optimizer that takes the logical plan and translates it to a physical plan based on data locality and node resources
- A storage engine interface that has the ability to mount any type of storage, from a local hard drive to a distributed storage like HDFS

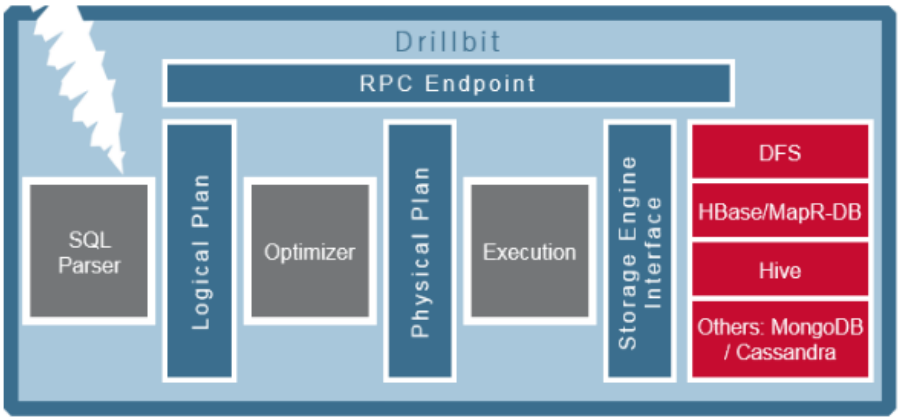


Figure 3.1: Drill components [36]

Drill in a cluster When using Drill as an SQL EE in a distributed storage, it is highly recommended to install a drillbit on all nodes. When set to perform MPP drill is dependent on Zookeeper to keep track of all the drillbits. When a query is submitted, the first node to respond to the initial request from Zookeeper becomes the *Foreman*. In Drill there is no master or slave hierarchy, and for every query submitted a new foreman is selected. Every drillbit contains all services and capabilities of Drill, which makes it highly scalable - just add another node and install Drill and you're done (after adding it to the Zookeeper Quorum). The foreman gets a list of available Drillbit nodes in the cluster from ZooKeeper, and determines the appropriate nodes to execute various query plan fragments to maximize data locality. This means that Apache Drill both performs the planning, distribution and execution of queries, only depending on Zookeeper to keep track of available nodes. Once a foreman has decided on the distribution of the query, the workload gets split downwards into fragments to the nodes, and the results get consolidated upwards until reaching the foreman, which then returns the results to the client.

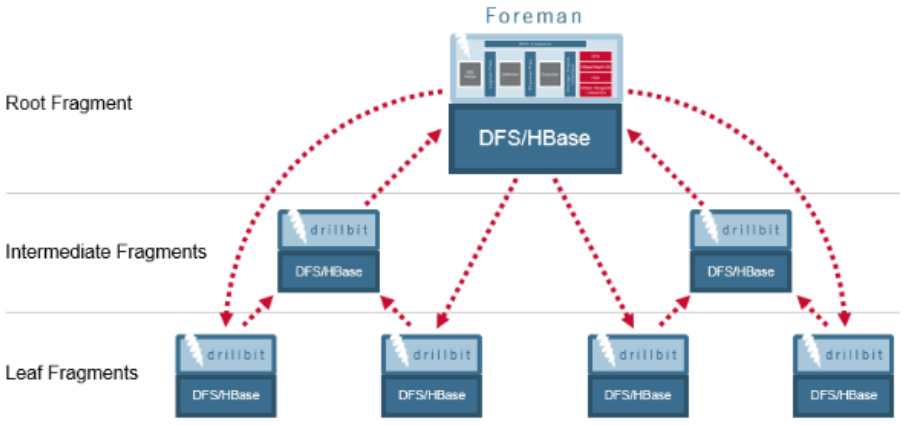


Figure 3.2: Drill parallel processing [36]

3.2.4 Drill ODBC Driver, and unixODBC

To be able to programatically perform queries to the Drill cluster we need an interface to connect to. This is done via a Open DataBase Connectivity (ODBC), that is installed on the NameNode of the Hadoop cluster. unixODBC is the self-proclaimed definitive standard for ODBC on non MS Windows platforms [6]. Once unixODBC was set up, the Drill ODBC Driver was installed on a arbitrary drillbit, although we chose the Hadoop NameNode for this as well. This then allows for setting up a Datasource referencing the Zookeeper Quorum which can then be contacted programmatically, interfacing Drill as if it were a standard SQL server / database.

3.2.5 pyodbc

There are plenty of ways to interface with an ODBC. We chose python as a preferred language because of familiarity. Therefore we chose to use pyodbc [7] to communicate with the ODBC, allowing a fully functional interface against our Drill configuration. pyodbc is a well established open source module, implementing the DB API 2.0 specification which is an API defined to encourage similarity between the Python modules that are used to access databases. Defining the Drill ODBC Driver as a datasource, which again points to the Zookeeper Quorum, proved effortless and fast.

3.3 Genetic algorithms

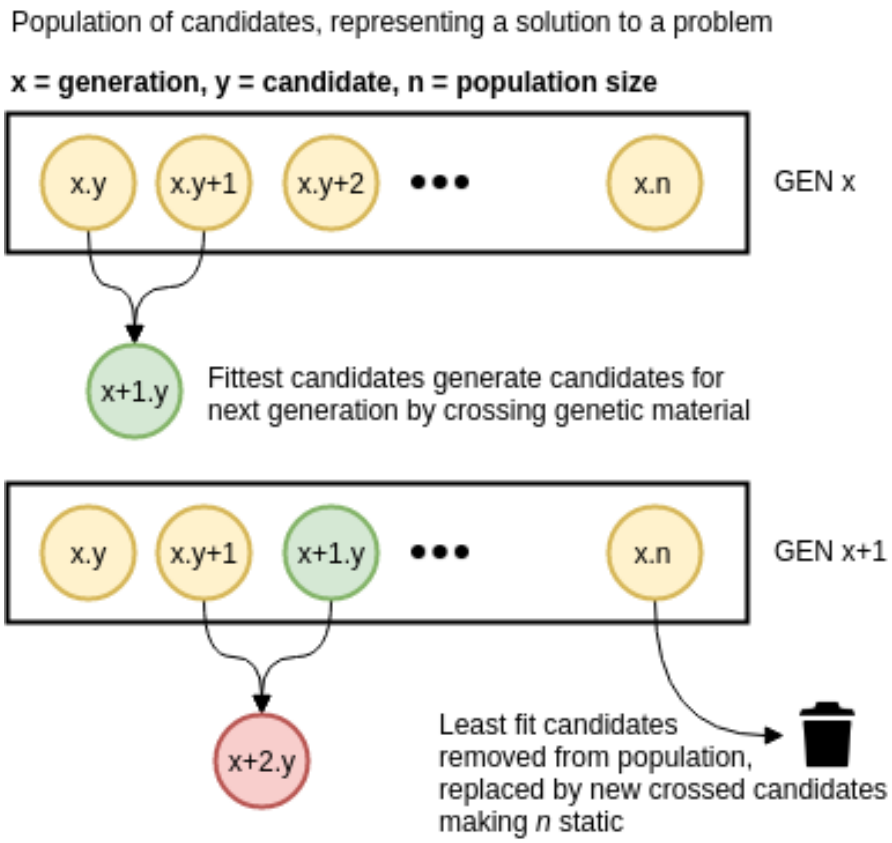


Figure 3.3: Basic concepts of GA

Genetic algorithms are higher level procedures for generating high-quality configurations / solutions to problems with a wide range of parameters, typically where exhaustive searches are infeasible. The procedure starts out with a population of n candidates, each representing a set of pseudo-randomly generated values for the respective parameters (properties) that will yield a result for the given problem. Once each candidate has attempted to solve the problem by applying their properties to it, a fitness score is given and compared amongst them. The fitness score represents how good their solution to the problem was, and lets us pick out the best candidates among our population. After the optimal candidates of a generation is chosen, a new generation is generated, inheriting the properties of the previous best candidates. This process repeats until a stop criterion is reached and the properties of the candidate is considered an optimal solution. The number of generations to be generated, and the number of parent candidates from which the next generation is generated from are tweakable parameters.

3.3.1 NASA applying genetic algorithms

A very famous application of genetic algorithms is the high-performance antenna NASA made, that actually flew in their Space Technology 5 (ST5) mission [15]. In NASAs case, they needed to make a highly receptive antenna with very small physical dimensions. It could have any number of branches, each going in any arbitrary direction. Because of these seemingly infinite possible configurations for the antenna, applying genetic algorithms to gradually evolve a design by testing randomly generated populations and combining the best traits from each candidate, proved to be a great way of solving this. *"(...) the current practice of designing antennas by hand is severely limited because it is both time and labor intensive and requires a significant amount of domain knowledge, evolutionary algorithms can be used to search the design space and automatically find novel antenna designs that are more effective than would otherwise be developed"* [15].

3.3.2 Limitations of genetic algorithms

Fitness function

At first glance it sounds like applying genetic algorithm techniques to any problem would yield an optimal solution, with relative ease. The problem though, is the fitness function for determining the candidate solution. As the complexity of the problem scales in size the search space for the fitness function will end up being too big to complete in a reasonable time. For instance if one single candidate evaluation took days, performed for the whole population, across generations, it could end up taking months completing just one single optimization. In those cases machine learning approaches or simply manual labor through technical and domain specific knowledge would prove to be more efficient.

Reaching the optimal solution

The only way to evaluate a solution is to compare it against other solutions within the population, all of which have are derived from random mutations. Therefore it is impossible to know whether a truly optimal solution is reached, or if one more generation of mutations will prove to give a better result. In practice a stop criterion for the candidate generation is therefore needed, where a solution is considered to be optimal.

3.4 Population based incremental learning

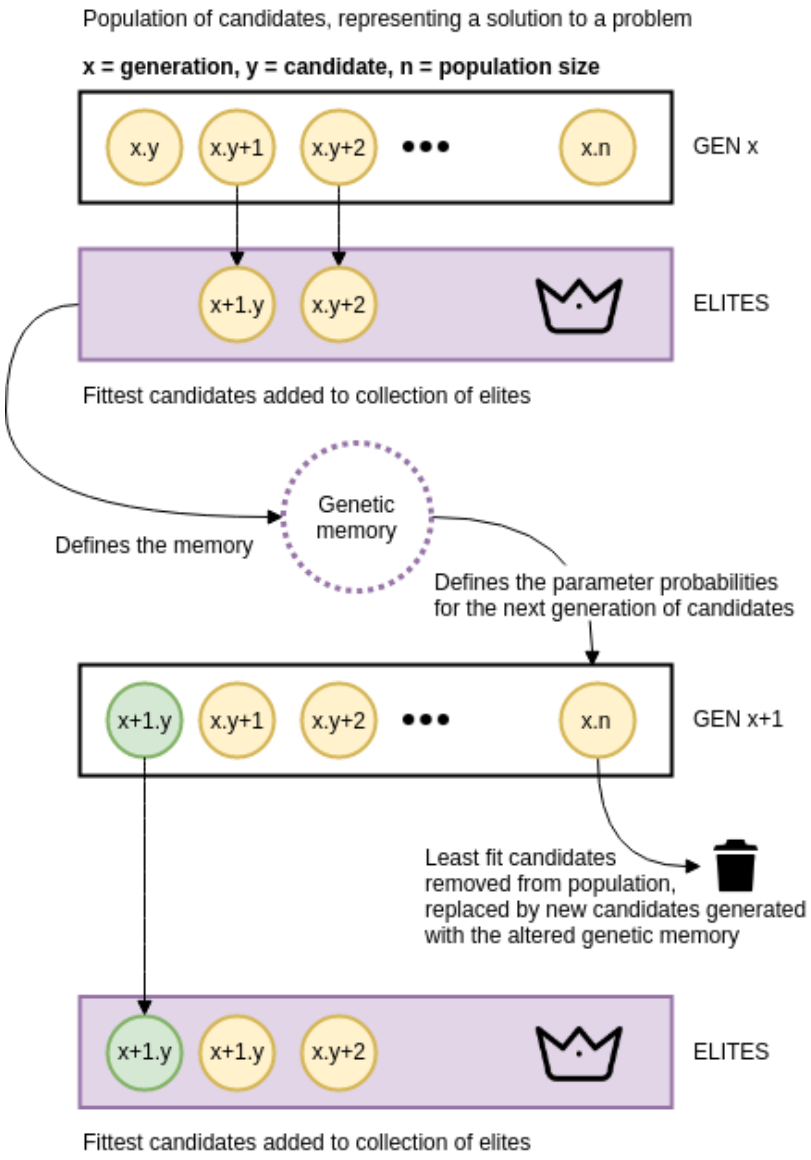


Figure 3.4: Basic concepts of PBIL

The distinction between PBIL and the general GA lies in where the evolution happens. GA generally focus on evolving individual candidates, and crossing fit parents to produce extra fit children, each of them with a chance of mutation. Across several generations convergence is expected. PBIL on the other hand evolves the whole population on a higher level, where the probability vector for parameter generation is shifted towards a considered favorable state. So instead of generating new candidates defined as a crossover function of two

parents, the fittest candidates affect a shared genotype that each candidate is generated from. This allows for a much clearer convergence of properties, where you end up in a state where every parameter has a (close to) 100% chance to be generated. This approach was first proposed back in 1994 by Baluja, where he found that *"The results achieved by PBIL are more accurate and are attained faster than a standard genetic algorithm, both in terms of the number of evaluations performed and the clock speed. The gains in clock speed were achieved because of the simplicity of the algorithm"* [28].

3.4.1 Advantage of PBIL

The main advantage PBIL supplies for this problem is the avoidance of unnecessary mutation on key properties, and generally faster convergence. In the case of Apache Drill there are a number of boolean properties that have big impacts on performance, i.e "planner broadcast join" that based on setting can result in a query that is *"(...) substantially cheaper"* [17]. The impact of flipping this boolean property is based on the type of query being done, the skewness of data locality and cluster size, but for a specific type of query in a set environment the property will always have a strictly optimal value. In PBIL this will become apparent early, and the chance of this value being anything else than the optimal one is likely decreased instantly after first generation run, and in every subsequent run. Typically this will result in a snowballing effect, where crucial parameters quickly get favored, further increasing the chance of them being selected, accelerating the selection process. Further more PBIL allows for complex values as genetic material instead of the usual binary representation, making it easier to implement linear parameters. A shared genetic memory for the population also matches well with the method presented in this thesis - of measuring performance of the candidate parameters on the SQL EE with an authority object. This is further detailed in the method chapter, judge.

3.4.2 Limitation of PBIL

The main limitation of PBIL is that each candidate property needs a preset of valid values. Thus finding the optimal solution is still based widely on the amount of options, and the effectiveness of them, defined by the developer of the algorithm. Naturally - in terms of boolean properties there lies no challenge. However when linear parameters need to be set, i.e properties with ranges from zero to infinity, the developer must define a set of valid options existing within that range. In the context of Apache Drill there are default rule of thumb values for each property, so the list of valid options for each property is generated around the default. Another drawback of PBIL lies in the risk of premature convergence. This can however be mitigated by effective mutation functions that ensure diversity.

3.5 Related literature and theoretical focus

3.5.1 Performance tuning MapReduce as a Research Field

There is a ton of research to be read about MapReduce optimization, trying to tune map- and reduce-slots, and improve the inherent job tracker. Looking at a few essential papers within this field shows a general consensus that tuning default performance parameters in a variety of environments will lead to 10-30% performance increase, which naturally results in considerable cost savings. For instance Min Li et. al. could report that "*Our results using a real implementation on a representative 19-node cluster show that dynamic performance tuning can effectively improve MapReduce application performance by up to 30% compared to the default configuration used in YARN*" [1]. Furthermore, manually tuning these clusters are too demanding for most businesses to even consider, requiring both deep technical and domain-specific insight. These findings further maintain our vision of bringing auto-tuning to the Apache Drill framework, which we believe to be the next natural step forwards, even paradigm-defining, for big data handling.

3.5.2 Mapreduce optimization

Gunther

Gunther evaluated methods for optimizing Hadoop configurations using machine learning and cost-based models, but found them inadequate for automatic tuning. Thus they introduced a search-based approach with genetic algorithms, designed to identify crucial parameters to reach near-optimal perfomance of the cluster [2]. This paper tells us that genetic algorithms as an approach to performance tune big data clusters already is a proven method. However, the scope is set to Hadoop as an out-of-the-box enterprise solution, whereas we take the next step within the schema-on-the-fly paradigm of Apache Drill.

mrOnline

"MapReduce job parameter configuration significantly impacts application performance, yet extant implementations place the bur- den of tuning the parameters on application programmers. This is not ideal, especially because the application developers may not have the system-level expertise and information needed to select the best configuration. Consequently, the system is utilized inefficiently which leads to degraded application performance" [1].

3.5.3 Other Hadoop SQL engines

There are plenty of SQL-on-hadoop engines available right now, like CitusDB, Impala, Concurrent Lingual, Hadapt, InfiniDB, JethroData, MammothDB, Apache Drill, MemSQL or Pivotal HawQ, and the amount of independently developed engines speaks volumes of the interest in this field. Why Apache Drill was chosen in this thesis is detailed in the Why Drill section, but the fact that Drill is (at the time of this writing) the only schema-free engine is a major selling point. However Impala and Spark are often considered as alternatives in similar environments.

3.5.4 Impala

Impala, developed by Apache is an opensource SQL engine that was designed to bring parallel DBMS technology to the Hadoop environment [29]. The main idea is shared between Drill and Impala where they both set up daemons on each node of a distributed cluster, to perform MPP (Massively Parallel Processing) for data reads, circumventing MapReduce. Impala is made for Big Data Analyics, priding itself in having *"order-of-magnitude faster performance than Hive"* [30].

3.5.5 Spark

Apache also developed Spark, a popular open-source platform for large-scale data processing that is well-suited for iterative machine learning tasks [31]. Spark is built with Hive, and runs very similarly to MapReduce but also has extended functionality. The key difference is that Spark keeps things in memory, while MapReduce keeps shuffling data in and out of disk. This allows Spark to facilitate machine learning algorithms on top of big data clusters, reading and processing data in the same workflow.

3.6 Presentation of domain where technology is used

The field of big data analytics is now becoming nigh impossible to ignore for enterprises dealing with customer information. Within this field Hadoop is currently the biggest platform. SiliconANGLE wrote a 5 year future broadcast in 2012, stating that *"Hadoop-MapReduce solution [will] be the de-facto industry standard for business intelligence and projects a 58% compound annual growth rate"* [34]. Global Knowledge Training highlights some of the reasons why Hadoop is seeing such success and adoption rate *"It's becoming clear that the open-source Apache Hadoop platform changes the economics and dynamics of large-scale data analytics due to its scalability, cost effectiveness, flexibility, and built-in fault tolerance. It makes possible the massive parallel computing that today's data analysis requires"* [33]. Further more BMC Software lists industries and sectors where Hadoop is currently being utilized to gain a competitive advantage, increase customer satisfaction or even improve citizen health:

- Financial services companies use analytics to assess risk, build investment models, and create trading algorithms; Hadoop has been used to help build and run those applications [3].
- Retailers use it to help analyze structured and unstructured data to better understand and serve their customers [3].
- In the asset-intensive energy industry Hadoop-powered analytics are used for predictive maintenance, with input from Internet of Things (IoT) devices feeding data into big data programs [3].
- Telecommunications companies can adapt all the aforementioned use cases. For example, they can use Hadoop-powered analytics to execute predictive maintenance on their infrastructure. Big data analytics can also plan efficient network paths and recommend optimal locations for new cell towers

or other network expansion. To support customer-facing operations telcos can analyze customer behavior and billing statements to inform new service offerings [3].

- There are numerous public sector programs, ranging from anticipating and preventing disease outbreaks to crunching numbers to catch tax cheats [3]. Hadoop is used in these and other big data programs because it is effective, scalable, and is well supported by large vendor and user communities [3].
- Hadoop is a de facto standard in big data [3].

Now it may seem like the Hadoop platform is an all-encompassing technology when businesses wants to deal with big data. There are however some bleaker recent reports stating that Hadoop adoption is going slower than previously predicted. A 2015 Gartner press release stated that *"[the] future demand for Hadoop looks fairly anemic"* [35]. The reasons businesses aren't adopting Hadoop as a big data framework however was fairly coherent, and is also highlighted in this thesis. It is too technically demanding to set up effectively. Companies that were reluctant or hesitant with Hadoop adoption cited skills shortage and user-unfriendliness as reasons for not thinking about Hadoop [35]. This is the exact reason why self optimizing frameworks like the one introduced in this thesis are needed. Making the Hadoop platform more approachable through a "out-of-the-box" mindset could potentially severely lower the threshold for taking a leap into big data. Through the methods presented in this thesis, we try to directly tackle this problem.

Chapter 4

Method

4.1 Infrastructure

To simulate a enterprise environment we set up a cluster consisting of four nodes, each with the same spec - 4 VCPUs and 8GB RAM. These were all VMs in OpenStack, each running Ubuntu 17.10. The cluster is also listed in the appendix, called the shared cluster The gateway has a outward-facing network interface for SSH access, and from there each of the other nodes can be managed. They all had Apache Drill installed, and were set up as a Hadoop cluster running HDFS between them. All data that Drill uses to read for tests are replicated three times in the cluster, ensuring parallel processing. The infrastructure stack

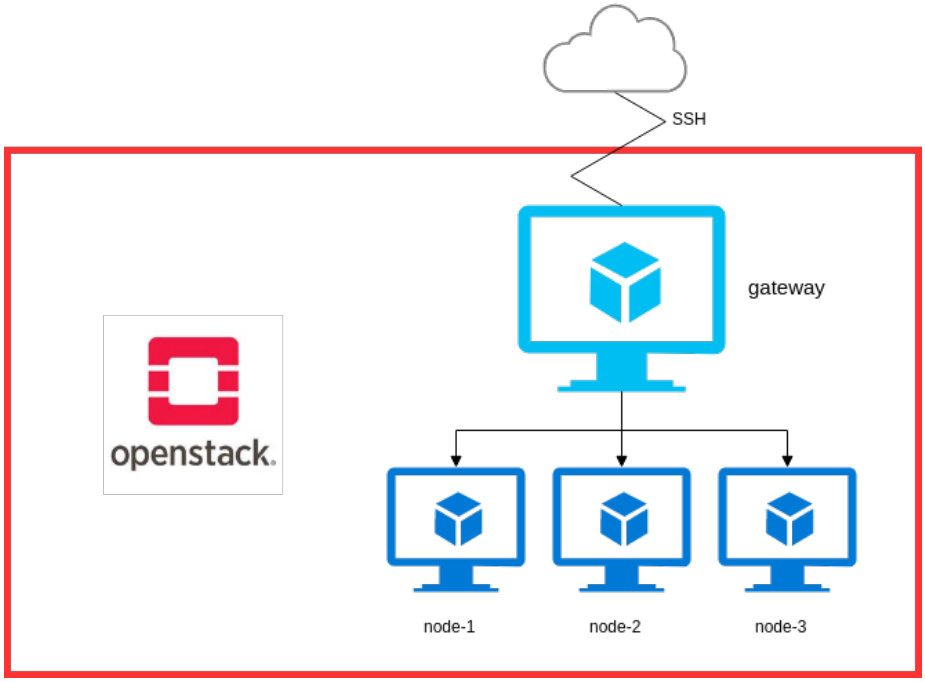


Figure 4.1: Cluster architecture

consists of the technologies listed in the technology section. All technologies are installed on all machines, except for the ODBC drivers, that are exclusively on the gateway. During a drill exection, any node can act as the orchestrator, named the *foreman*. On each machine there is a user named *hadoop*, that runs all the services needed to keep the cluster operational, like the zookeeper server and the drillbits. The gateway also has a main user simply named *ubuntu*, that runs the tests and the algorithm. The drillbits are made available through a ODBC DSN. The services are as follows:

- Gateway
  - NameNode (hdfs)
  - SecondaryNameNode (hdfs)
  - QuorumPeerMain (zookeeper)
  - Drillbit (drill)
  - Jps
- node-1, node-2, node-3
  - DataNode (hdfs)
  - QuorumPeerMain (zookeeper)
  - Drillbit (drill)
  - Jps

The Jps service is simply the Java Virtual Machine Process Status Tool, used to monitor the status of all the other services listed.

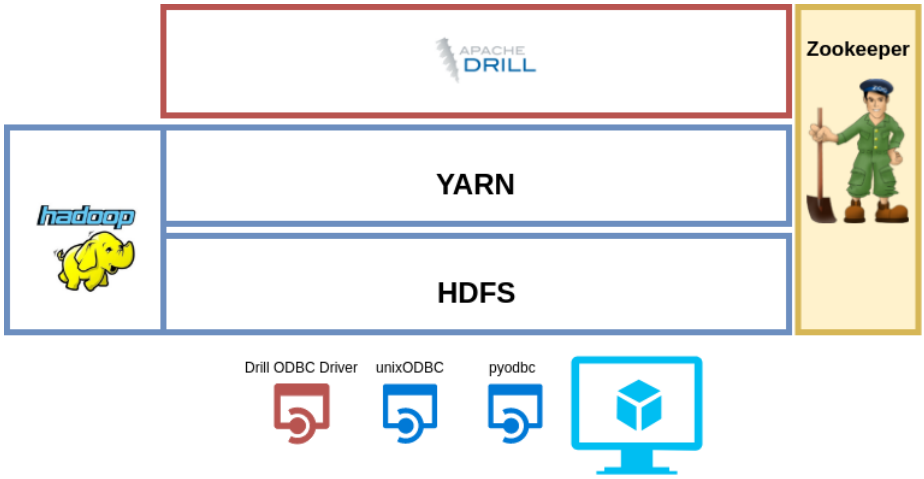


Figure 4.2: Technology stack

4.2 Optimization system overview

The overall design of the self optimizer is object oriented, where the judge is the overarching authority in a tree structure. The judge holds the memory and all candidates, and the candidates hold the settings. This makes data easy to keep track of in the optimization process, as everything can be read from the judge.

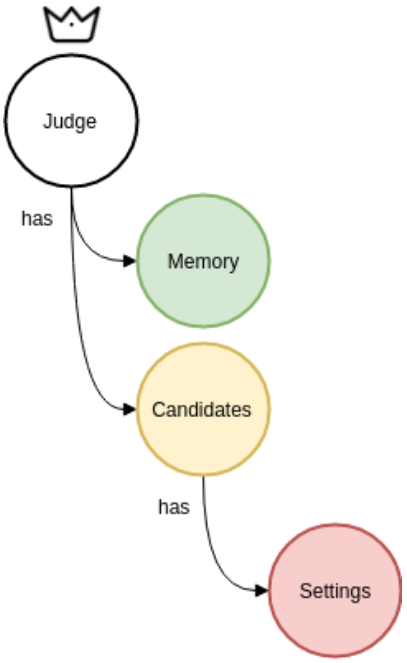


Figure 4.3: Hierarchy of objects

4.2.1 Judge

When initializing a optimization process the first crucial thing to get created is the judge. The judge extracts all of the information contained within the configuration file, to construct itself, the memory object, a default candidate and an initial population.

Judge operations

Memory The first thing the Judge needs to do, apart from declaring various variables to the itself, is generating a memory object. Without the memory the judge can't initialize any candidates, as they are dependent on the memory to define probability vectors for parameter creations. The global genetic material that defines PBIL is stored in the memory, or rather **is the memory**. The memory is further explained in the next subsection.

Default candidate To define goals for the fitness function a candidate is created, with all default values. This candidate also comes into play at the end of a optimization process, to verify solution. In the beginning, the default candidate is simply run the same way as all future solution candidates will, and metrics for mean, maximum and minimum TTE is stored within. The goal is then set at 50% reduction from the default candidates mean time, giving a somewhat unrealistic goal to ensure that we strive towards the most cost effective solution. In the end we run a gauntlet between the default candidate and the winner (optimal solution), where we measure the averages - to ensure that there's no uncontrolled variance that led to the result.

Population Based on a parameter from the configuration file, the judge generates an entire generation of n candidates representing a soluion to the problem. For each generation the judge measures the fitness of all candidates and executes the weakest ones. Then the strongest ones get added to the elite club, which is a set of candidates we call the elites. The elites are the ones that refine the memory, incrementally adapting it and reinforcing good solutions. After the memory has been altered by the newly added elites (and potentially elites that were added earlier), new candidates get added to the population to replace the ones that got executed. The configuration file can drastically alter the way the population behaves, by changing values for operations that are done on each generation:

- The size of the population
- The number of elites to add to the club each generation
- The number of executions / replacements each generation
- The size of the elite club
  - To keep the memory alterations based on fresh new solutions, we want to evict older members. Some versions of PBIL does not include evictions - but for achieving convergence, evictions are crucial.

4.2.2 Memory

The memory is the heart of this entire system. It provides a centralized brain that makes decisions based on historic data, and incrementally adapts it's knowledge as new data is being generated. While it can be considered a machine learning technique, the memory has a truly finite usage within each run, as it is meant to reach a state of convergence. More classic incremental learning schemes are continuously adapting, attempting to for instance predict the stock market. In our case however we simply want to reach one final solution and then end the run. For each run the memory starts out fresh, with a uniformly distributed probability vector, ensuring that changes in i.e cluster topology or data skewness will be taken into consideration when optimizing. Thus it can be valuable to execute one full optimization run each time a big data environment changes, or new data sources get added. If the memory were to store historic genetic material, then certain probability vectors would be skewed in favor of previously strong parameters, that may not be beneficial for current or future runs.

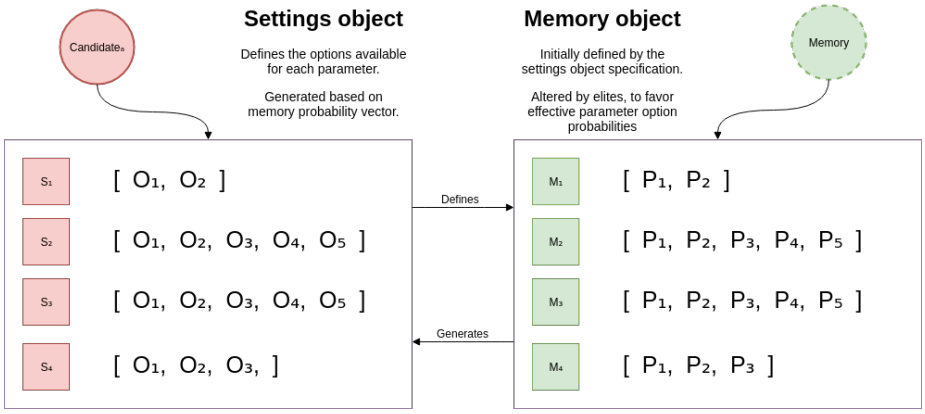
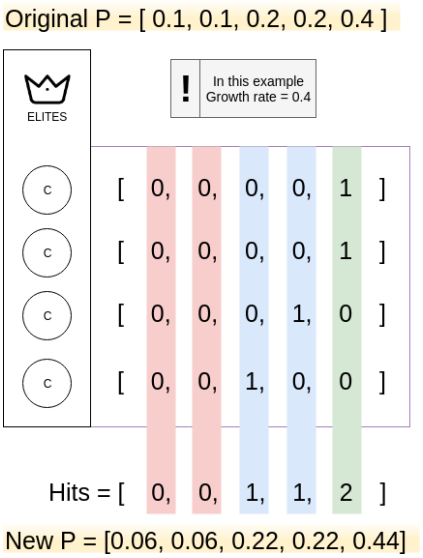


Figure 4.4: The reliance between a candidates' setting object, and the judges' centralized memory.

Probability vector algorithm When altering the memory, we iterate over each elite in the elite club, and view their setting parameters. Each option for a given parameter in the candidate is represented as a list, containing any type of data. The list can be as short or as long as the user likes - it will automatically get a uniform probability distribution in the memory and be eligible for optimization. For each parameter the candidates' setting object contains, there is a mirrored probability vector in memory. When the alterations take place, we check the candidate parameter value and locate it in the list of options, and then return a binary list of hits. For instance if the parameter is boolean the logic would look like this:

1. Parameter value is **True**
2. Parameter options are [**True,False**]
3. Binary representation will be [**1,0**]

Figure 4.5: How the memory gets altered by the elites



Once this binary option conversion is done for all candidates, we sum up the hits of each option and get a single list for all accumulated hits, as seen in figure 4.5. Finally we can apply the mathematical formula to alter the probability vector, using the accumulated hits list, denoted as:

$$P_{i,j}(t) = P_{i,j}(t) + \lambda \left(\frac{\mu_{i,j(t)}}{|\mu|} - P_{i,j}(t) \right) \tag{4.1}$$

$$P = probability, i = parameter, j = option, \mu = elite, t = time$$

Reaching convergence After each memory alteration the memory object checks itself, and each parameter within the setting object it holds. It then counts each parameter, and whether it has passed the convergence threshold. Once all parameters has passed the threshold convergence is reached. This method of evaluating convergence is however very susceptible to parameter agnosticism, leaving the algorithm running for a long time trying to reach convergence on parameters that the elites do not agree on, due to the nature of those parameters being superfluous. The time cost here led to the development of accelerated convergence.

Accelerated convergence Once the total memory convergence value has reached 90% of the convergence threshold, the memory enters a state of accelerated convergence. In this state the growth factor of the genetic algorithm increases by 0.2. This is due to parameter agnosticism in many queries, and especially in less complex queries or environments. This is further detailed in the discussion section, parameter agnostic queries.

4.2.3 Extended memory function - progress estimation

The memory object is also responsible for a novel approach to timing genetic algorithm process times. The single biggest drawback of using genetic algorithm techniques for complex problems is execution time, and especially as it's commonly impossible to predict. Still we are able to create two individual progress bars, to give users some idea of the time optimization should take.

```
##############################
########## Optimizing #########
##############################


|###########                                  |29.09 % of critical parameters converged

|###################################          |74.90 % total convergence


############ Testing generation 54 ############
Accelerated convergency active
Running Candidate 50.0
Running Candidate 51.1
Running Candidate 52.1
Running Candidate 52.4
Running Candidate 53.3
Running Candidate 54.0
Running Candidate 54.1
Running Candidate 54.2
Running Candidate 54.3
Running Candidate 54.4
```

The first bar estimates when critical parameters are converged. This is done by tracking convergence values over time, and using a linear regression model to predict when the convergence threshold should hit. The linear regression model fails at one point, because the prediction starts to inverse - i.e it predicts that total convergence value should pass convergence threshold on generation #60, while we're already at generation #65. This then indicates strongly that elite consensus is missing, and the remaining values put forth begins to flatline, meaning only arbitrary parameters remain to be settled. The second bar represents total convergence and is a simple function of current convergence * 100 - considering how current convergence is a number between 0-1.

Algorithm 1 Critical convergence estimation

```
# Estimate when convergence of critical parameters are done
cl = list of all convergence values for generations passed
il = list of cl indexes
model = linear regression model
fit model(cl,il)
convergence = model.predict(#CONVERGENCE_THRESHOLD)
diff = convergence - il[-1]

target = max(diff)
percentage = ((diff*(-1) + target) / target) * 100

# Construct progress bar based on percentage

USER DEFINED CONSTANTS:
CONVERGENCE_THRESHOLD: Threshold for when a parameter is considered to be converged
```

Algorithm 1 has been tested thoroughly and proven to be an effective estimation for critical convergence, with one such test graph displayed in graph 4.6. Further details on elite consensus, critical vs arbitrary parameters and parameter agnostic queries can be read in the discussion chapter, Examples of agnosticism from stable cluster

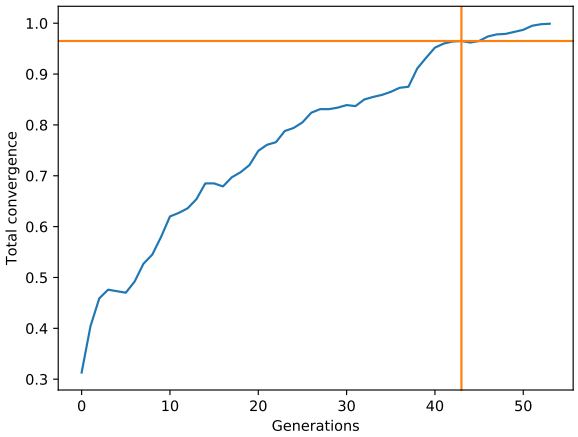


Figure 4.6: Data taken from a demo run where model estimated that critical parameters were converged at generation #43, shown here as the orange cross section. This can also be observed as the point in time where convergence first flatlined, even slightly dipping, meaning elite consensus was missing going forward. Since this is from a demo run, the run count was set to the lowest possible setting, thus converging at a very early stage at just above generation #50.

4.2.4 Candidate

Candidates don't hold much information on their own, as most of it is stored in a setting object, which the candidate holds on to. However the candidate has some crucial functions, like how to run queries with their setting object, how to mutate and how to measure self performance.

Run To get data on whether or not a particular set of settings will be an effective solution in the current environment, we need to run some tests - here simply defined as runs. To complete a run in this system the candidate is dependent on a data source, and information on how many runs to do, in order to measure the TTE. In a cluster where resources are shared among many users, or multiple queries are ran concurrently on the same quorum, the optimal solution need to take this into consideration. Therefore runs are done in two stages, one for concurrency testing, and another for sequential testing. Both the data source and the number of runs to complete are defined by the configuration file. The data source defined here is a ODBC DSN, but for future work this can be easily altered to point to a JDBC DSN or other similar sources. The number of runs defined by the configuration file is actually doubled in runtime, as it defines both the amount of concurrent runs in addition to the amount of sequential runs. This means that defining this setting to the absolute minimum of 1 run, results in two runs - one in the concurrent stage and one in the sequential stage. Performing only a single run would leave too much up to circumstance, in regards to the available resources in the cluster and random error, thus a minimum of two runs is needed to measure performance - although any number higher than this is advised. Further technical details on how the runs are performed can be read in the testing section

4.2.5 Mutation

To mitigate the issue of premature convergence a mutation function is in place, to mutate new candidates at a set probability defined by the configuration file. After a candidate is created, it may be chosen for mutation. The value that defines the probability for a candidate to be chosen also decides the probability for individual parameters to mutate within the candidates setting object. There is also a resistance value in place, to avoid too heavy mutations, potentially resulting in a useless candidate at later stages, which simply increases the time cost of the algorithm. The resistance value starts at 0, and increases via the resistance growth value set in the configuration file. Each time a parameter is mutated within a candidate, the candidate becomes more genetically resistant. It works by linearly reducing the mutation chance for individually chosen candidates. By default this reduction is 1% per mutation. Increasing the resistance growth would result in more stable candidates, more purely defined by memory, but gives less variations - so this needs to be balanced.

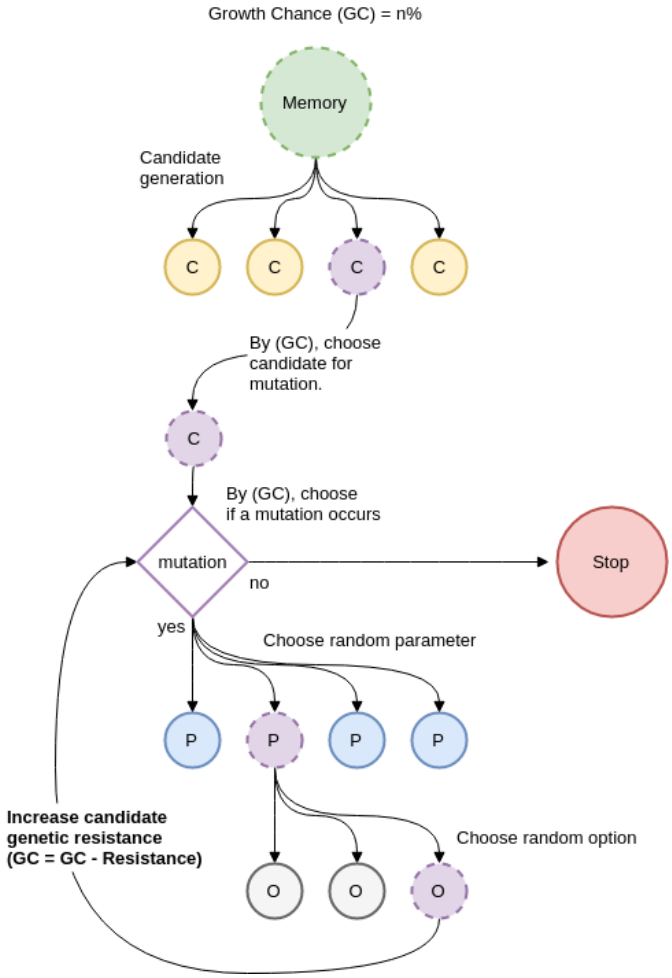


Figure 4.7: Mutation function flowchart

The mutation is entirely isolated from the memory, and thus chooses a new option for the parameter randomly, within a candidate. If a mutation is specifically advantageous in the given environment the candidate will go on to become an elite, thus altering the memory with its genetic material.

PBIL Mutation vs GAPBIL Mutation

While the original implementation of the PBIL mutation function seen in algorithm 2 focused on mutation of the collective probability vector, the GAPBIL approach seen in algorithm 3 mutates individual candidates - hoping to produce a diverse set of elites.

Algorithm 2 Original PBIL mutation

```
# Mutate Probability Vector
for #LENGTH
    if (random (0,1) < MUT_PROBABILITY)
        Pl <- Pl * (1.0 - MUT_SHIFT) + random (0.0 or 1.0) * (MUT_SHIFT)

USER DEFINED CONSTANTS:
LENGTH: length of encoded solution
MUT_PROBABILITY: probability of mutation occurring in each position
MUT_SHIFT: amount for mutation to affect the probability vector
```

Algorithm 3 GAPBIL mutation

```
# Mutate Candidate Parameters
if random (0,1) < #GROWTH_CHANCE:
resistance = 0
while (mutation occurs)
    if random < #GROWTH_CHANCE - resistance:
        chosen_parameter = random (0, length(parameters)-1)
        chosen_parameter.option = random available option
        resistance += #RESISTANCE_GROWTH / 10
    else:
        stop mutation, end while loop

 USER DEFINED CONSTANTS:
 GROWTH_CHANCE: probability of choosing candidate and/or parameter for mutation
 RESISTANCE_GROWTH: amount of mutation resistance for candidate, for each mutation occurance
```

4.2.6 Setting

The setting object is simply a collection of session setting for Apache Drill to apply before running queries. An innovation of GAPBIL is the ability to easily implement linear parameters as opposed to the binary representation of genetic material that GA or PBIL uses. As such, each setting is gathered from the documentation from the Apache team, and experimented on to see which ones affect TTE. The full list of parameters can be viewed in appendix A.
Table A.1 shows what parameters where used, that where able to be altered in session. Table A.2 shows what parameters had to be cut, due to them only being able to be changed out of session, or them causing segmentation faults.

4.2.7 Configuration file

This thesis focuses on removing the barrier for adoption of big data frameworks. As such one might find it ironic that a configuration file is supplied, that severely impact the performance of the optimization algorithm. However it is expected that the supplied default values set for the algorithm is sufficiently effective for any environment where this is run. With the default values there is a fine balance between execution time of the algorithm, and the margin for how fine grained a optimal solution should be. In addition to this there are very simple guide lines provided, regarding how big the cluster is, how many cores each node has, and how much time to be spent for accuracy sake. All the parameters can be found in appendix B.

4.2.8 Algorithm overview

Flowchart Figure 4.8 shows a simplistic flowchart for how the entire system runs, from start to achieved convergence. Algorithm 4 shows brief pseudo code of the basic concepts of GAPBIL.

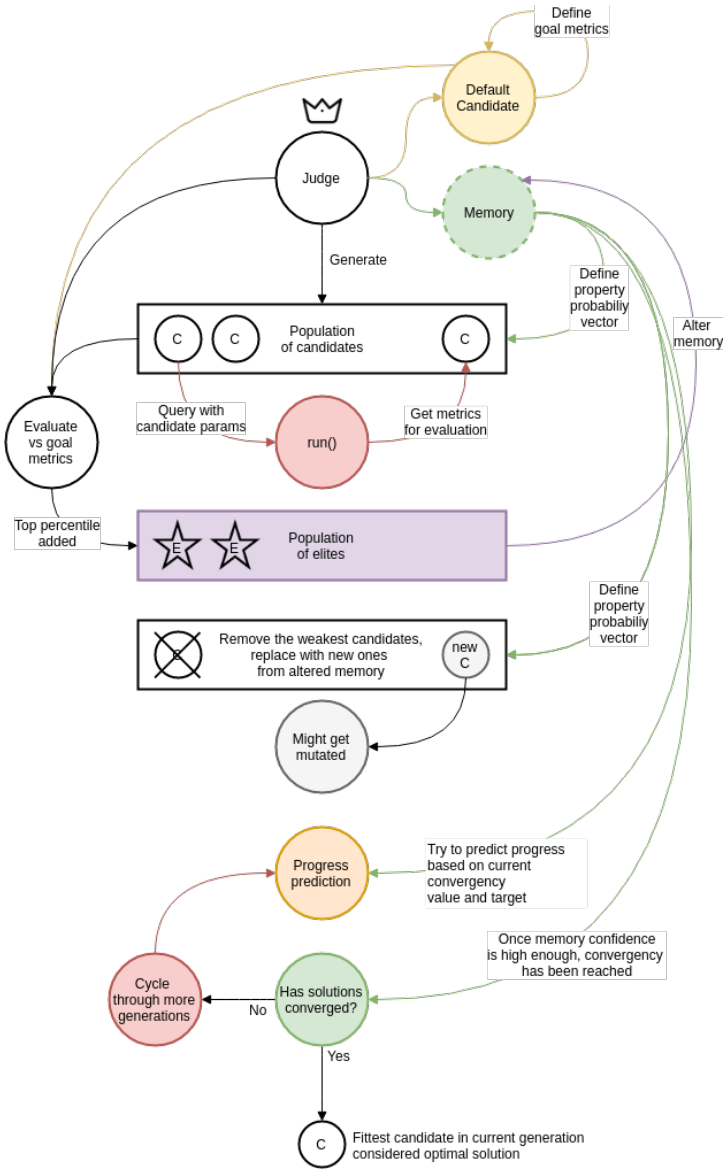


Figure 4.8: System overview

Algorithm 4 Basic GAPBIL algorithm

```
# Generate default candidate
default <- generate solution according to collective genetic memory
goal <- defined by performance of default solution

# Generate candidate population
for #POPULATION
    candidate_{i} <- generate solution according to collective genetic memory
    evaluation_{i} <- measure solution performance against goal

# Perform generation cycle
for #POPULATION
    elites_{t} <- add #ELITE candidates from population
    if elites_{t} > #ELITE_CLUB
        evict oldest suplerflous elites
population <- remove #CYCLE weakest
execute probability vector algorithm with elites_{t}
generate and add #CYCLE candidates according to altered genetic memory to population

# Check for convergence
for parameters in memory
    count all parameters with probability > #CONVERGENCE_THRESHOLD
if converged
    End algorithm, get fittest candidate and test optimal solution for assurance

USER DEFINED CONSTANTS:
POPULATION_SIZE: amount of candidates in population
ELITE_CYCLE: amount of elites to add each generation cycle
ELITE_CLUB_SIZE: amount of elites kept in memory for memory alterations
CANDIDATE_CYCLE: amount of candidates to replace each generation cycle
CONVERGENCE_THRESHOLD: threshold for considering a parameter converged
```

4.3 Testing

4.3.1 Dataset

The dataset used to test the algorithm on the clusters come from Yelp, specifically from their dataset challenge [32]. The data was gathered as JSON files, containing reviews, stores, customer visits and the like.

4.3.2 Query

A query combining joins, grouping and sub queries was built to increase the complexity the Drill SQL EE had to handle. Further more the Drill specific notation KVGEN and FLATTEN was used for Key Value Generation after Flattening a complex json value to represent it as a table:

```
SELECT Business, Days, Total_Checkins
FROM (SELECT Business, Days, SUM(tot1.Checkins.`value`) as Total_Checkins
FROM (SELECT tot.name as Business, tot.timvals.key as Days,
FLATTEN(KVGEN(tot.timvals.`value`)) as Checkins
FROM (SELECT FLATTEN(KVGEN(tbl.`time`)) as timvals, business.name
FROM (SELECT * FROM `/user/hadoop/drill/yelp_data/checkin.json` LIMIT 1000) as tbl
JOIN `/user/hadoop/drill/yelp_data/business.json` as business
ON business.business_id=tbl.business_id) as tot) as tot1
GROUP BY Business,Days) WHERE Total_Checkins>150 ORDER BY Business,Total_Checkins
```

4.3.3 Run flowchart

Performing queries on the cluster programatically through python requires many layers of middleware. The Python language uses pyodbc, to speak to ODBC, that has defined a DSN on the Zookeeper Quorum, to access the Drillbits that reside on HDFS, as seen in figure 4.9. Every time a query is run we measure the time it takes, and store it in a global list. This is important in regards to concurrency, as the threads themselves cannot return values to the main function, but rather append their values to a fixed index in the result list.

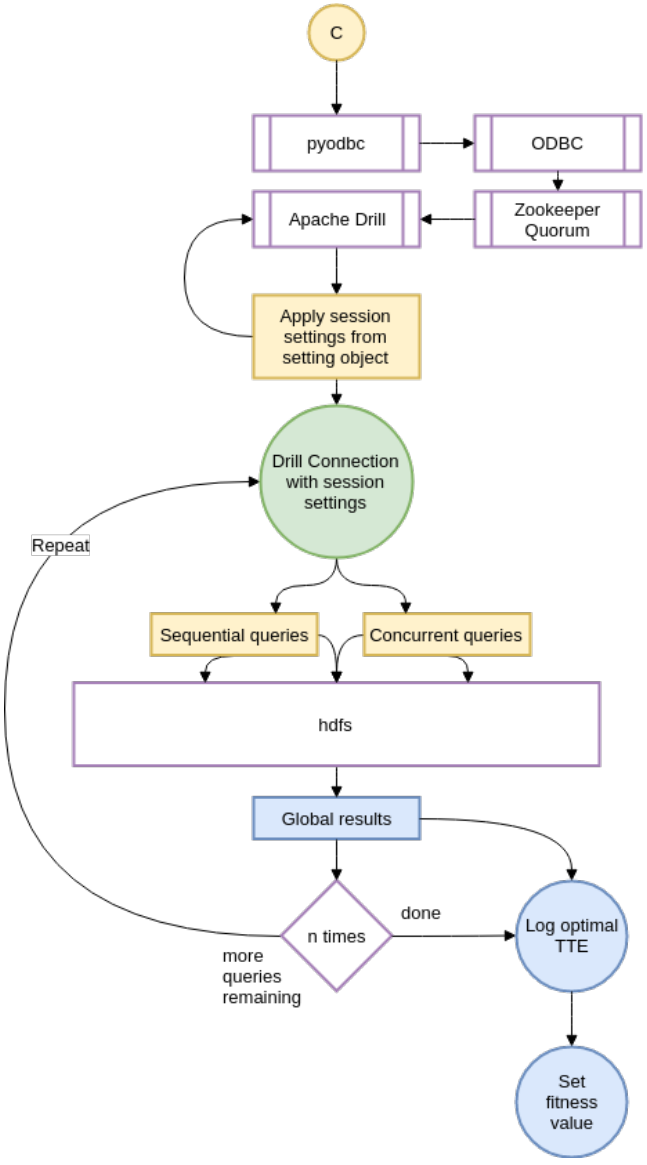


Figure 4.9: All the technological layers that we pass during a single query

4.3.4 Data collection

When executing a run from a candidate we preferably do many queries before measuring fitness. The SQL and the data is always the same between all candidates, runs and queries. In addition to this we only take the optimal TTE into consideration when measuring a candidates effectiveness. This counts for all candidates generated from memory as well as the default candidate, as shown in figure 4.9. This ensures they all compete on even ground, and no noise or variance effects the result. When performing a optimization, a user can either input a custom name for the run, or simply execute it without arguments, causing the name to be *default*. After the scores are settled, all the collected data is stored in the logging folder under the chosen name, stored as a JSON file named as current timestamp. Data collected is:

- how many generations the optimization took
- average, mean and optimal query times for all elites
- average, mean and optimal query times for default candidate
- average, mean and optimal query times for default candidate and optimal solution during assurance runs
- Cost saving for all candidate solutions vs default benchmark run
- Total convergence values for all generations

All graphs presented in the results chapter 5 are genereated based on this JSON file. In addition to the JSON file, a file named log.txt is generated for each successful optimization, containing the optimal solution parameters in relation to the default solution- looking like:

```
#### RUN: 2018.04.16.19.44.55
Setting                                          Default value     Optimal value  Confidence
-----------------------------------------------  ----------------  ----------------  ------------
planner.memory.enable_memory_estimation                  0                 1            100.00%
exec.queue.enable                                        0                 0            100.00%
planner.broadcast_factor                                 1                 0.75         100.00%
planner.broadcast_threshold                            1e+07             5e+06          100.00%
planner.slice_target                                1000              5000              100.00%
planner.width.max_per_query                         1000               100              100.00%
exec.min_hash_table_size                           65536             32768              100.00%
exec.max_hash_table_size                              1.07374e+09       1.07374e+09     100.00%
exec.queue.large                                       10                 2.5            100.00%
exec.queue.small                                      100                50              99.98%
exec.queue.threshold                                   3e+07             4.5e+07         100.00%
exec.queue.timeout_millis                          300000            450000              100.00%
planner.memory.max_query_memory_per_node              2.14748e+09       2.14748e+09     100.00%
planner.width.max_per_node                            11                 9              100.00%
planner.add_producer_consumer                          0                 0              100.00%
planner.enable_hashjoin_swap                           1                 0              100.00%
planner.enable_mergejoin                               1                 0              100.00%
planner.filter.max_selectivity_estimate_factor         1                 1              100.00%
planner.filter.min_selectivity_estimate_factor         0                 0.4            100.00%
planner.join.hash_join_swap_margin_factor             10                12.5            100.00%
planner.join.row_count_estimate_factor                 1                 0.75           100.00%
planner.memory.average_field_width                     8                14              100.00%
planner.memory.hash_agg_table_factor                   1.1               1.5            100.00%
planner.memory.hash_join_table_factor                  1.1               1.1            100.00%
planner.memory.non_blocking_operators_memory          64               128              100.00%
planner.partitioner_sender_max_threads                 8                 4              100.00%
planner.nestedloopjoin_factor                        100               125              100.00%
planner.producer_consumer_queue_size                  10                50              97.58%
store.text.estimated_row_size_bytes                  100               400              99.99%
store.json.all_text_mode                               0                 0              100%
store.json.read_numbers_as_double                      0                 0              100%
```

4.3.5 Solution assurance

As mentioned earlier we only use optimal TTE to measure default candidate and solutions, to reduce noise. However another layer of noise reduction was added at the end of the algorithm process, to ensure that no individual variance or noise caused the solution to be picked, even with purely optimal TTEs. This is named the solution assurance stage, running several runs, 10 by default, where the considered optimal candidate competes against the default candidate. Furthermore to make sure no cluster instability skews the assurance in any candidates favor, the default candidate and optimal candidate take turns doing runs - meaning the default solution is applied every nth run.

4.3.6 Adapting data collection

Because of the object oriented approach of this algorithm, where the judge owns all the data, and gets passed along as an object for storing the data, it is really simple to adapt what to store. Every kind of metric belongs to the judge, and the judge is last object that we interact with before writing to disk. Therefore future work, or other use cases for GAPBIL should have no problem with metric analysis, even on complex structures like nested objects or lists.

Chapter 5

Results

5.1 What we display

Graphs overview Following the reasoning laid out in the method chapter, we've ran several experiments. As the experiments were carried out, the data collected was slowly refined until we reached a final version where every source of noise was gone, and the graphical output to the user was meaningful and easy to read. Naturally then, all data showed in this chapter is collected from experiments performed on the stable cluster. The main metric we measure to decide upon algorithm effectiveness is the *Minimum query* value - which tests both the default candidate and the optimal solution under optimal conditions. Metrics for TTE, convergence and fitness for experiments are shown. In addition to this a graph for the solution assurance stage is included for each experiment, to clearly visualize the default candidate performance vs the optimal solution from GAPBIL.

Cost saving graphs Data collection for the cost saving graphs, are gathered as the run evolves. It is based on each elite from its respective generation, vs the default candidates first run, which sets the benchmark goal. This first default run often turns out to be a high TTE outlier in Drill, which means the cost saving looks to be a lot higher than the solution assurance stage proves. This is further discussed in the discussion chapter, drill cache.

Mean calculations for elite runs There are averages plotted in the elite run graphs, that are smoothed considerably for better visualizing the results. The averages are calculated by first collecting all elite runs for a generation (by default 3), and averaging their results so we get a generation TTE. Then we add this generation TTE to a list, times the amount of elites we measured **??**. Finally we perform gaussian smoothing on the generated list twice **??**. This makes the mean line smooth and easy to read.

5.2 Run configurations

Configurations used were the same for all experiments, namely:

```
{

"POPULATION_SIZE": 10,
"GROWTH_FACTOR" : 0.4,
"GROWTH_CHANCE" : 0.2,
"RESISTANCE" : 0.1,

"CONVERGENCE_THRESHOLD" : 0.95,
"MAX_GENERATIONS" : 200,
"CONCURRENT_RUNS" : 2,

"REPLACEMENTS_EACH_RUN" : 5,
"ELITES_EACH_RUN" : 3,
"ELITE_CLUB_SIZE" : 7,

"DATA_SOURCE" : "drill64",
"NODES" : 4,
"CORES" : 4,
"NODE_LEAST_MEMORY" : 2147483648,

"DISPLAY_OUTPUT" : false,
"LOGGING_FOLDER" : "/home/ubuntu/runlogs/",
"RUN_NAME" : "default"

}
```

Because of the very low memory specifications in the stable cluster, we had to settle for a concurrency value of 2, meaning 4 queries per candidate run. This is in actuality lower than advised for the algorithm to run effectively and produce certain solutions. Even so we could see an improvement in all experiments done during the solution assurance stage, proving that the algorithm performs well even under less than optimal settings, in low end environments.

5.3 Highlighted experiments

TODO: display graphs properly

5.3.1 Experiment 1

```
Convergence threshold reached at generation 128


Results:
Metric           Optimal solution    Default  Overhead saved
-------------  ------------------  ---------  ----------------
Optimal query              9.20229     9.3086  1.14%
```

Experiment 1 achieved 1.14% reduction in optimal query times, in 128 generations before converging. This is one of the runs discussed in the section about accelerated convergence, because of the long time it took for elites to reach consensus on parameter options, as shown in figure 5.1. It initially rises tremendously in total convergence up until generation 20, before spending over 100 generations to settle for the remaining parameters. Viewing the cost saving in graph 5.2 in relation to the convergence graph 5.1 we can see why convergence after generation 20 is hard to reach, as the cost saving flatlines. Figure 5.3 shows the optimal query times for each elite gathered from each generation. It curves downwards nicely, but it is clear the first few elites generated were bad solutions, with TTE as high as 9.8. Each generation had 3 elites added, meaning that according to the fitness and convergence charts, the elites TTE should balance out at around generation 20, which here will be seen as elite #60. This is confirmed by looking at the spread, in which the mean line revolves around 9.15 from around elite 60# onward. Finally figure 5.4 shows the solution assurance runs. Our optimal solution repeatedly performs better in the 10 subsequent runs performed during the solution assurance stage.

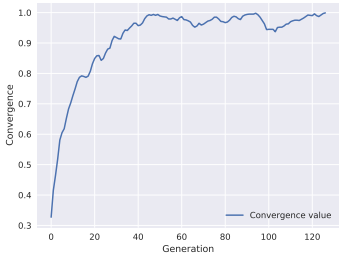


Figure 5.1: Experiment 1: Total probability convergence over generations

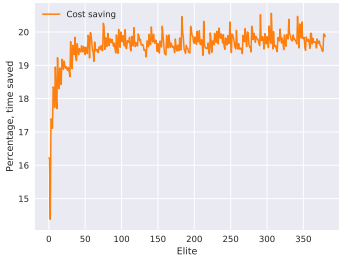


Figure 5.2: Experiment 1: Cost saving of optimal elite run vs default benchmark

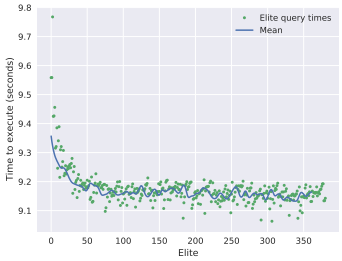


Figure 5.3: Experiment 1: Elite run graph, showing elites optimal TTE over generations

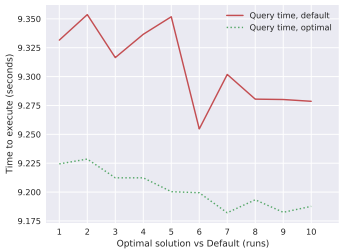


Figure 5.4: Experiment 1: Solution assurance stage, showing optimal solution vs default candidate

5.3.2 Experiment 2

```
Convergence threshold reached at generation 138


Results:
Metric           Optimal solution    Default  Overhead saved
-------------  ------------------  ---------  ----------------
Optimal query             9.15923    9.32121  1.74%
```

Experiment 2 achieved 1.74% reduction in optimal query times, in 138 generations before converging, being the experiment that took the longest to converge. In this experiment the first goal defining default run did not spike as high as in the other experiments, leading to a cost saving graph 5.6 that shortly dips below 0%. Furthermore there is a tremendous dip in cost saving at elite #77, which coincides with the elite run graph 5.7 around the same time. This drop happens after the cost saving flatline, so it is regarded as a random error, possibly due to network latency. The elite run graph shows a couple of high TTEs within the span of the first 10 generations but drops sharply, getting one major outlier spike, before oscillating around 9.1. Viewing the solution assurance graph 5.8 shows that the optimal solution repeatedly beats the default candidate, even when experiencing variations and oscillations.

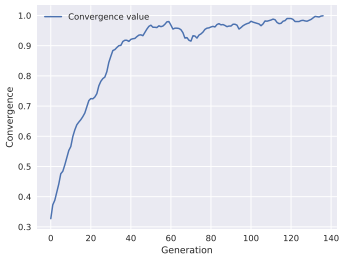


Figure 5.5: Experiment 2: Total probability convergence over generations

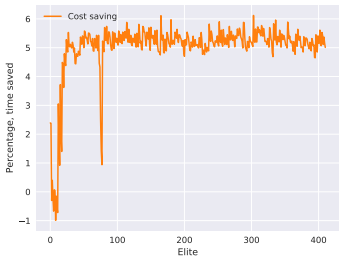


Figure 5.6: Experiment 2: Cost saving of optimal elite run vs default benchmark

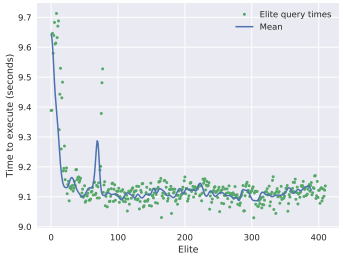


Figure 5.7: Experiment 2: Elite run graph, showing elites optimal TTE over generations

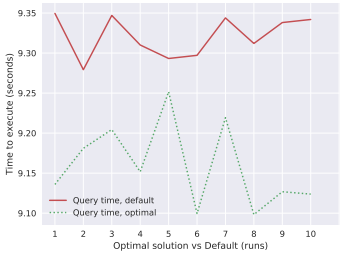


Figure 5.8: Experiment 2: Solution assurance stage, showing optimal solution vs default candidate

5.3.3 Experiment 3

```
Convergence threshold reached at generation 90


Results:
Metric           Optimal solution    Default  Overhead saved
-------------  ------------------  ---------  ----------------
Optimal query              9.45051    9.61736  1.73%
```

Experiment 3 achieved 1.73% reduction in optimal query times in just 90 generations, before converging, being the fastest optimization measured. The convergence graph 5.9 never flattens out or oscillates as badly as the previous runs, rather continuing on a upward trend until reaching total convergence. The cost saving graph 5.10 also shows a small jump in the end of the run, that can be read in relation to the elite run graph 5.11 as a drop in mean. During the solution assurance stage shown in graph 5.12 There was a sharp drop in the default candidates TTE at run 8, possibly being an overall outlier - meaning that the total cost saving for this candidate solution could average out a bit higher given a higher run count for solution assurance.

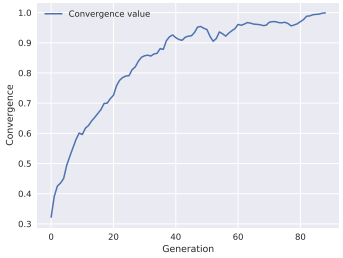


Figure 5.9: Experiment 3: Total probability convergence over generations

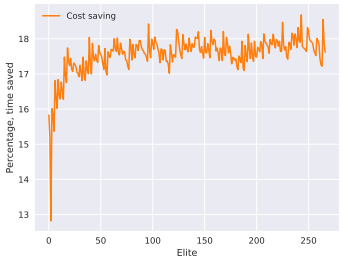


Figure 5.10: Experiment 3: Cost saving of optimal elite run vs default benchmark

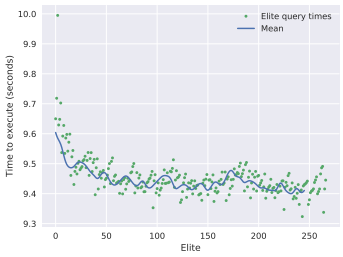


Figure 5.11: Experiment 3: Elite run graph, showing elites optimal TTE over generations

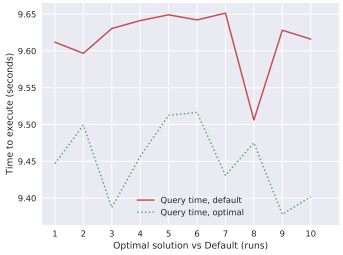


Figure 5.12: Experiment 3: Solution assurance stage, showing optimal solution vs default candidate

5.4 Aggregated results

Chapter 6

Discussion

6.1 The drill cache

As mentioned previously, a candidate run consists of two stages - one concurrent stage and one sequential stage. A single run on a single candidate will always perform at least two sequential queries. All candidates in a population are run in sequence, one after the other whilst measuring their fitness. Still we see that the first run of the optimization process, running the default candidate, is measured to have higher TTE than subsequent runs when performing solution assurance in the end - if the run count is low. This is caused by drill populating a distributed cache, which the default candidate will utilize fully in subsequent runs, but not in the goal defining run. This source of deviance from the true performance of the default candidate is however irrelevant in the grand scheme, as the goal could be an arbitrary value to beat for the generated candidates. After all the algorithm does not end when a goal is reached, rather when convergence is achieved. When performing solution assurance in the end the true performance gets measured against the considered optimal solution, and a fair assessment of both candidates' TTE is made, proving effect - or lack thereof.

6.2 Parameter agnostic queries

6.2.1 Tiny queries, non-complex problems

Running a very simple query in the cluster environments we set up, on a small json file of a couple thousand lines - took drill around 0.2 seconds. Attempting to optimize for such queries was completely futile, as the only bottleneck for such a simple query is pure hard drive reading speeds. This led to every randomized set of settings to be considered equally good, which again means that the default candidate is equally good. No optimal setting can be discovered, and convergence of the algorithm is simply an act of chance.

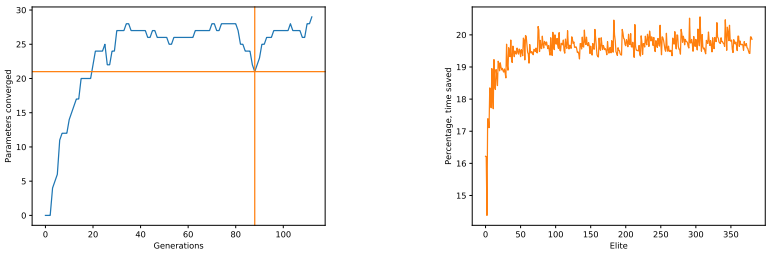
6.2.2 Complex queries

As queries get more difficult to plan, the data set gets bigger, the cluster gets more heterogeneous and the data source more complex - the impact of per-

formance tuning increases. As discussed previously drill has different phases when executing a query, like the logical and the physical planning phases. The planning phases heavily depend on the configuration settings and performance tuning parameters to calculate how to carry on execution of a query. We discovered early on that in a given environment there were some parameters that heavily impacted the execution time, and some others that were completely arbitrary, or even ignored by drill in runtime. This led to the creation of the accelerated convergence state in the memory object, to increase probability vector growth once the critical parameters have been sorted out.

6.2.3 Examples of agnosticism from stable cluster

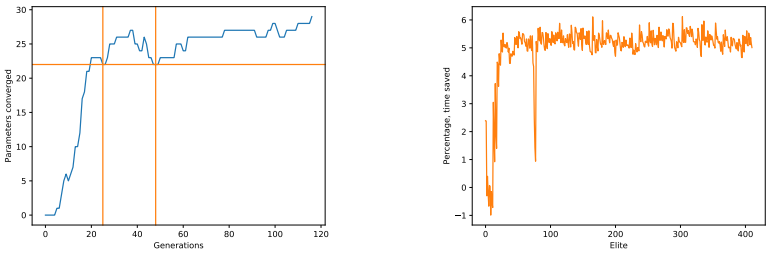
Runs often see total convergence reach around 0.8 - 0.9 and staying there for tens of generations, without ever increasing the fitness value. To further inspect the hypothesis of parameter agnosticism a graph of converged parameters over time was created, and a cutoff points of critical vs arbitrary parameters was charted out, as seen in figure 6.1. (*Keep in mind that this metric is different than the one used by the critical parameter estimation model seen in algorithm 1 which uses total average convergence values*) After 21 parameters reached convergence, we can see that there is no consensus among the elites on the remaining parameter options, leading total convergence to oscillate instead of steadily increase. In this chart we see one convergence drop down to its lowest of 21 once, 68 generations after first passing that point - before heading for total convergence.



(a) label 1 (b) label 2

Figure 6.1:
a) Blue line showing amount of parameters converged over time for a run. Orange lines showing lowest convergence point drop after steep growth (x88,y21).
b) Graph over elite fitness in the same run, clearly flatlining at the same time as elite consensus for remaining parameters becomes uncertain.

In another example shown in figure 6.2 there are two low drops to 22 converged parameters. In both graphs there is a overwhelming consensus among elites of what parameters result in the best solution, up until 21-22 parameters are converged. In both cases this initial elite consensus takes around 20 generations. From that point on around 100 generations are spent trying to reach consensus on the remaining 7-8 parameters. Keep in mind that one generation cycle can easily take around 15 min. Given 10 candidates in the population, times 10 queries per candidate, times 10 seconds per query results in 16.6 minutes per generation.



(a) label 1 (b) label 2

Figure 6.2:
a) Blue line showing amount of parameters converged over time for a run. Orange lines showing lowest convergence point drops after steep growth (x22,y25 and x22,y48).
b) Graph over elite fitness in the same run, clearly flatlining at the same time as elite consensus for remaining parameters becomes uncertain, except for one outlier which we disregard as random error.

After introducing accelerated convergence, we give the algorithm an extra push in the last 10% towards convergence, to make sure we don't waste time and resources on a random gamble of irrelevant parameters to the current environment and / or data source. Convergence still takes a lot of time in event of heavy parameter agnosticism, but cutting the algorithm off prematurely to force default options on the respective arbitrary parameters could lead to faulty configurations in edge cases, so we settled for speeding it up instead - with accelerated convergence.

6.3 Error sources

6.3.1 Drill planner inconsistency

First of all, drill is inherently a bit varied in terms of query times. Performing the same query twice in the same session will heavily leverage a shared cache amongst the drillbits, reducing query times by an average of 15%. Further more the planning phase contains some fuzzy logic and decision making probabilities, so it may not always result in the same execution stage. The cache issue is mitigated by letting every candidate start a new session, and run the same amount of queries in the session. The planning stage inconsistency is mitigated by running several queries in a single run, to make sure to take the average into consideration. The solutions to both these problems are however parameterized, so setting the amount of queries per run to a low number will lead to increasing impact of these factors as error sources.

6.3.2 Cluster instability

In the initial testing phase, results from runs varied widely in execution time. The cluster however, was a shared resource where many projects were competing for computing power. Even running only the default candidate for 50 generations, extreme spikes in TTE as high as 30% were seen, randomly distributed amongst the runs. When cluster instability gets added on top of drill planner inconsistency in a single query / run, it leads to an outlier of very high TTE. However with such variations it is hard to reach a sensible convergence for truly optimal solutions, considering how fitness scores then are products of both settings and circumstance. This led to a migration to a similar sized cluster of 4 nodes, but with a lower spec, where the resources where allocated and could not be competed for, hereby called the stable cluster. The stable cluster mitigated the variations and gave good grounds for studying the result of the algorithm.

6.3.3 Cluster size

Enterprise environments where optimization techniques like this would be most beneficial, are also the hardest ones to simulate. Increasing the amount of nodes and the amount of resources would add to the complexity, mitigating the optimization problem of parameter agnosticism.

6.4 Proving effect

After analyzing our data it is clear that the algorithm works as intended - reducing the overall TTE. It is proven across several solution assurance runs that the average cost saving of the suggested optimal solution is worthwhile, averaging out at 1,53% for the stable cluster, for the highlighted experiments. For the shared cluster cost saving is inconclusive, because of the heavy variance in TTE caused by the shared resources, leading to faulty results.

6.5 Cut configuration parameters

Time constraints and choice of implemented algorithm caused some relevant parameters to be cut from the optimization process, making them default values in all candidates. There were a total of six cut parameters, viewable in the appendix, cut parameters. The two execution queue memory parameters could not be set in-session, as they were exclusively system settings. This meant that we would need to restart apache for each candidate, to apply their system settings, causing massive overhead and added complexity. The four remaining ones all define what type of aggregation and join functions should be available. Changing these parameters led to segmentation faults while running candidates, and they then had to be cut from the settings list. Given more time, some deeper study and insight could be gained to understand the segmentation faults, and perhaps build systems to handle them.

Chapter 7

Further work

7.1 Homogeneous clusters

In this thesis both the clusters set up consisted of a homogeneous set of nodes. It is expected that a heterogeneous cluster will have added complexity to the logical and physical planner on the drillbits, further increasing the value of optimization. This is a high priority effort for future work, as it accurately represents a real world use case.

7.1.1 Load profiles

It should be infeasible for one configuration to solve all problems. After all, the concept of optimization takes great consideration to its specific context. Expecting to solve all aspects of a complex problem with one remedy leads to a candidate being a jack of all trades but master of none. Hence the solution should be tailored to a specific context, by letting the algorithm aggressively tune the system to work in its current state. To make sure this algorithm performs this way, testing needs to be done on different kinds of data sets, with different kinds of queries.

7.2 Single node setups

It would be interesting to gather data from a single node laptop setup for running queries in drill. The way data sources work in this thesis through pyodbc meant that it would add too much development time to adapt for local hard drive runs, either in the way of configuring a hard drive location as an ODBC DSN, or altering the way we programatically run drill queries.

7.3 Support for the cut parameters

As mentioned earlier there are a couple of interesting parameters that was cut from the development process in this thesis, listed in the appendix. To handle the segmentation faults potentially caused by hash- and aggregate join disabling, one could possibly access drill through a different source than pyodbc, to create

an appropriate wrapper around the runs and catch the errors. To handle the system settings, a more complex run needs to be set up, to restart the cluster and check for readiness before querying it for each candidate - this may ultimately be infeasible in terms of optimization time.

Chapter 8

Conclusion

8.1 Innovation - GAPBIL

8.1.1 Inspiration

This thesis started out using a simple Genetic Algorithm for the candidate solutions. Further in the development process PBIL was implemented, as a better fit for the object oriented authority based system made, with the judge at the top. Even still some alterations to the algorithm had to be made to reach the intended goal, and thus PBIL was not satisfactory. This led to the creation of a new novel algorithm named GAPBIL.

8.1.2 Genetic flexibility

GAPBIL accepts any type of genetic material a user would want, from numbers and strings to complex values like lists and matrices. The parameters can even be objects themselves. This is all handled by using memory object held by the judge, that mirrors the parameters held by the candidates. The flexibility in genetic material is a considerable improvement over traditional genetic algorithms, which purely allow for binary genetic values.

8.1.3 Mutation

Genetic algorithms mutate candidates. PBIL mutates the collective probability vectors. GAPBIL mutates on candidate level, potentially allowing an advantageous mutated candidate to reach elite status, thus altering the collective probability vector. In such a way GAPBIL combines the methods presented in both GA and PBIL, hence the name.

8.2 Goal

8.2.1 Self optimizing system

GAPBIL is by every definition a self optimizing system. A user of the algorithm simply need to define how many parameters a setting object should have, and what options they should contain, to solve a given problem. The rest is up to the judge.

8.2.2 Apache drill performance tuning

No set value for how major the impact of performance tuning a drill cluster needed to be to be considered a success. With an average of 3% cost saving in the testbed environment on a static query upon a large JSON dataset, we consider it a success. If the optimizer were to be run in more complex environment on more diverse datasets we are confident that the cost saving will be at least as much. Due to time constraints however, we need to conclude on the results at hand.

8.3 Most valuable contribution

The most valuable contribution of this thesis lies in the algorithm GAPBIL, which can be used to solve any complex problem, improving upon traditional genetic and evolutionary algorithms with a major increase in genetic material flexibility.

Bibliography

[1] Li, M., Zeng, L., Meng, S., Tan, J., Zhang, L., Butt, A. R., & Fuller, N. (2014, June). *Mronline: Mapreduce online performance tuning. In Proceedings of the 23rd international symposium on High-performance parallel and distributed computing (pp. 165-176).* ACM.

[2] Liao, G., Datta, K., & Willke, T. L. (2013, August). *Gunther: Search-based auto-tuning of mapreduce.* In European Conference on Parallel Processing (pp. 406-419). Springer Berlin Heidelberg.

[3] BMC Software, Inc (2018, January) *http://www.bmc.com/guides/hadoop-examples.html*

[4] Apache Software Foundation (January 2018) *https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html*

[5] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., & Babu, S. (2011, January). *Starfish: A Self-tuning System for Big Data Analytics.* In Cidr (Vol. 11, No. 2011, pp. 261-272).

[6] Nick Gorham (2018, February) *http://www.unixodbc.org/*

[7] Michael Kleehammer (2018, February) *https://github.com/mkleehammer/pyodbc*

[8] White, T. (2012). *Hadoop: The definitive guide.* " O'Reilly Media, Inc.".

[9] Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). *Dremel: interactive analysis of web-scale datasets.* Proceedings of the VLDB Endowment, 3(1-2), 330-339.

[10] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). *The Google file system* (Vol. 37, No. 5, pp. 29-43). ACM.

[11] Dean, J., & Ghemawat, S. (2008). *MapReduce: simplified data processing on large clusters.* Communications of the ACM, 51(1), 107-113.

[12] Google LLC (2018, February) *https://cloud.google.com/customers/*

[13] Modis (2018, February) *http://www.modis.com/it-insights/infographics/top-it-jobs-of-2018/*

[14] Johannessen, R., Yazidi, A., & Feng, B. (2017, April). *Hadoop MapReduce scheduling paradigms. In Cloud Computing and Big Data Analysis* (ICCCBDA), 2017 IEEE 2nd International Conference on (pp. 175-179). IEEE.

[15] Hornby, G., Globus, A., Linden, D., & Lohn, J. (2006). *Automated antenna design with evolutionary algorithms.* In Space 2006 (p. 7242).

[16] Apache Software Foundation (2018, January) *https://drill.apache.org/docs/analyzing-the-yelp-academic-dataset/*

[17] Apache Software Foundation (2018, January) *https://drill.apache.org/docs/join-planning-guidelines/*

[18] Apache Software Foundation (2018, February) *https://drill.apache.org/docs/release-notes/*

[19] Ireland, C., Bowers, D., Newton, M., & Waugh, K. (2009, March). *A classification of object-relational impedance mismatch.* In Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA'09. First International Conference on (pp. 36-43). IEEE.

[20] 2018 BetterCloud Monitor (2018, February) *https://www.bettercloud.com/monitor/real-time-enterprise-messaging-comparison-data/*

[21] Stack Exchange Inc, posted by user: Thoth (2018, February) *https://stackoverflow.com/questions/42403229/mysql-database-with-thousands-of-tables*

[22] Gantz, J., & Reinsel, D. (2012). *The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east.* IDC iView: IDC Analyze the future, 2007, 1-16

[23] Porter, M. E., & Heppelmann, J. E. (2015). *How smart, connected products are transforming companies.* Harvard Business Review, 93(10), 96-114.

[24] Bendre, M. R., & Thool, V. R. (2016). *Analytics, challenges and applications in big data environment: a survey.* Journal of Management Analytics, 3(3), 206-239.

[25] Huang, G., He, J., Chi, C. H., Zhou, W., & Zhang, Y. (2015, June). *A Data as a Product Model for Future Consumption of Big Stream Data in Clouds.* In 2015 IEEE International Conference on Services Computing (SCC), (pp. 256-263). IEEE.

[26] Statista, Inc (2018, February) *https://www.statista.com/statistics/ + [266206/googles-annual-global-revenue/, 277229/facebooks-annual-revenue-and-net-income/]*

[27] Github user Edlich, curated list (2018, February) *http://nosql-database.org/*

[28] Baluja, S. (1994). *Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning* (No. CMU-CS-94-163). Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science.

[29] Bittorf, M. K. A. B. V., Bobrovytsky, T., Erickson, C. C. A. C. J., Hecht, M. G. D., Kuff, M. J. I. J. L., Leblang, D. K. A., ... & Yoder, M. M. (2015). *Impala: A modern, open-source SQL engine for Hadoop.* In Proceedings of the 7th Biennial Conference on Innovative Data Systems Research.

[30] Apache Software Foundation (2018, March) *https://impala.apache.org/overview.html*

[31] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Xin, D. (2016). *Mllib: Machine learning in apache spark.* The Journal of Machine Learning Research, 17(1), 1235-1241.

[32] Yelp Inc. Yelp (2018, March) *https://www.yelp.com/dataset/challenge*

[33] Global Knowledge Training LLC. (2018, March)
Big Data and Apache Hadoop Adoption: Key Challenges and Rewards *https://www.globalknowledge.com/us-en/content/articles/big-data-and-apache-hadoop-adoption/*

[34] SiliconANGLE Media, Inc. (2018, March) *https://siliconangle.com/blog/2012/08/01/hadoop-big-data-reports-future-growth/*

[35] Gartner, Inc, Gartner Survey Highlights Challenges to Hadoop Adoption *https://www.gartner.com/newsroom/id/3051717* STAMFORD, Conn., May 13, 2015

[36] MapR Technologies, Inc. (2018, March) *https://mapr.com/blog/apache-drill-architecture-ultimate-guide/*

Appendix A

Cluster specs

Table A.1: Specs for the shared cluster

Specification	**Value**
Amount of nodes	4
Heterogeneous	No
OS	Ubuntu 17.10
VCPUs	4
RAM	8GB
HDD	80GB

Table A.2: Specs for the stable cluster

Specification	**Value**
Amount of nodes	4
Heterogeneous	No
OS	Ubuntu 16.04
VCPUs	2
RAM	2GB
HDD	15GB

Appendix B

System parameters

Table B.1: All parameters and options used by PBIL in this thesis

Name	Default value
planner.memory.enable_memory_estimation	False
exec.queue.enable	False
planner.broadcast_factor	1
planner.broadcast_threshold	10000000
planner.slice_target	1000
planner.width.max_per_query	1000
exec.min_hash_table_size	65536
exec.max_hash_table_size	1073741824
exec.queue.large	10
exec.queue.small	100
exec.queue.threshold	30000000
exec.queue.timeout_millis	300000
planner.memory.max_query_memory_per_node	max_query_memory
planner.width.max_per_node	calculated
planner.add_producer_consumer	False
planner.enable_hashjoin_swap	True
planner.enable_mergejoin	True
planner.filter.max_selectivity_estimate_factor	1
planner.filter.min_selectivity_estimate_factor	0
planner.join.hash_join_swap_margin_factor	10
planner.join.row_count_estimate_factor	1
planner.memory.average_field_width	8
planner.memory.hash_agg_table_factor	1.1
planner.memory.hash_join_table_factor	1.1
planner.memory.non_blocking_operators_memory	64
planner.partitioner_sender_max_threads	8
planner.nestedloopjoin_factor	100
planner.producer_consumer_queue_size	10
store.text.estimated_row_size_bytes	100

Table B.2: Parameters and options that was removed during development, but still is considered to have impact on performance of query execution

Name	**Default value**
planner.enable_multiphase_agg	True
planner.enable_broadcast_join	True
planner.enable_hashagg	True
planner.enable_hashjoin	True
exec.queue.memory_ratio	10
exec.queue.memory_reserve_ratio	0.2

Appendix C

Configuration parameters

Table C.1: Configuration parameters that define how the algorithm executes.

Name	**Default value**
POPULATION_SIZE	10
GROWTH_FACTOR	0.4
GROWTH_CHANCE	0.3
RESISTANCE	0.1
CONVERGENCE_THRESHOLD	0.95
MAX_GENERATIONS	200
CONCURRENT_RUNS	10
REPLACEMENTS_EACH_RUN	5
ELITES_EACH_RUN	3
ELITE_CLUB_SIZE	7
DATA_SOURCE	drill64
NODES	4
CORES	4
NODE_LEAST_MEMORY	2147483648
DISPLAY_OUTPUT	false
LOGGING_FOLDER	/home/ubuntu/runlogs/
RUN_NAME	default

Appendix D

Code