UiO : Department of Informatics
University of Oslo

Tuning of Elasticsearch Configuration

Parameter Optimization Through Simultaneous Perturbation Stochastic Approximation Algorithm

Mohamad Sobhie

Thesis submitted for the degree of Master in
Network and System Administration
30 credits

Department of Informatics
Faculty of mathematics and natural sciences

Autumn 2019

Tuning of Elasticsearch Configuration

Parameter Optimization Through Simultaneous Perturbation Stochastic Approximation Algorithm

Mohamad Sobhie

14th December 2019

Acknowledgement

I would like to thank my supervisors, Anis Yazidi and Hårek Haugerud, for the encouragement and advice they have provided throughout my time as their student.

I would like to also thank my parents for their love, caring, and prayers. I am very much thankful to my brother, Omar, who has always been there for me. I want to extend my gratitude to my close friends for their support throughout my two years of studies.

Abstract

By default, Elasticsearch configuration does not change while it receives data. However, when Elasticsearch stores a large amount of data over time, the default configuration becomes an obstacle in scaling for better performance. Besides, the machine that hosts Elasticsearch will have limitations on its specifications, like memory size. A solution to this problem is to tune the parameter configuration of Elasticsearch, which leads to achieving better performance. One way to tune parameters is by using Simultaneous Perturbation Stochastic Approximation. This report provides an implementation of optimizing Elasticsearch configuration parameters by observing the performance and automatically change the configuration to provide better performance. The used implementation relies on combining machine learning with ELasticsearch. Through this combination, Elasticsearch configuration can change its configuration parameters automatically without the need to reset the currently running instance of Elasticsearch. The results showed a good improvement in the number of inserted data and response time of the system.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

API Application Programmable Interface

CPU Central Processor Unit

DevOps Development and Operations

DSL Domain Specific Language

FDSA Finite difference stochastic approximation

JSON JavaScript Object Notation

JVM Java Virtual Machine

ML Machine Learning

NoSQL Not Only Scripted Query Language

PS Page Size

RAM Random Access Memory

RDBMS Relational Database Management System

SA Stochastic Approximation

SPSA Simultaneous Perturbation Stochastic Approximation

TTN Time To Notify

Chapter 1

Introduction

In this introductory chapter, the motivation for this project is presented with a problem statement and questions to resolve. The chapter starts by presenting the motivation, following the description of the problem statement, and finally, it presents the thesis outline.

1.1 Motivation

The amount of generated data is remarkably increasing every day, 3.8 billion people use the internet as of 2017. It is also estimated that 1.7 MB of data will be created for every person every second by 2020 [1]. Also, other data will be generated by servers to maintain their status and stored in files, which are referred to as log files. Log files can include different types of data, such as web requests from users, user activities, server events, etc. Log files are considered part of big data [2]. Due to the increase in the number of users and machines, there will be a large amount of data to analyze.

There are different aspects when discussing large data, one of these aspects is searching among big data. When it comes to searching methods, the taken time to fetch the right information crucially influences the quality of a search engine. Having big data has led to the need for good search engines in which information is quickly collected and is relevant to the searched input. Elasticsearch [3] is a search engine that became popular within the Development and Operations (DevOps) field, and also among many tech companies [4]. It can be combined with other tools that collect logs from servers and visualize them.

The traditional way of operating the development teams in organizations has led to less efficiency and more conflicts when new features or updates are pushed to production [5]. For this reason, a demanding need for filling the gap between development and operations has created the DevOps field, that is, a field in which operations, development, and quality assurance teams are unified [6]. This has made it faster to release new codes into production, and it also increased the quality of software systems [7].

Even though the current DevOps tools do not majorly rely on machine learning (ML), there has been some interest in applying machine learning into DevOps tools where the quality of software processes was enhanced [8]. This has brought attention to how to increase quality by applying optimization solutions. Elasticsearch configuration relies on several parameters, and this means that tuning the right parameters will give better results in terms of used resources and fast output, hence quality.

By combining ML and Elasticsearch, we can achieve better performance by tuning some parameters in Elasticsearch using ML. This project will handle different aspects of combining ML with Elasticsearch and will provide an algorithm to tune Elasticsearch in an efficient way.

1.2 Problem Statement

Running Elasticsearch with a good performance level depends on what server specification you have. However, there are other parameters that affect the performance of Elasticsearch. These parameters can provide fast searching or indexing if configured correctly. To do so, one has to adopt an optimization solution to have an optimal-like configuration. The following problem statement describes what this report will cover:

How to achieve a better Elasticsearch performance by applying Simultaneous Perturbation Stochastic Approximation algorithm while Elasticsearch cluster keeps on scaling

In order to achieve the above, it is important to understand what parameters to deal with and how to deal with them. Along with other considerations too, this report will answer the following questions:

- How to dynamically change Elasticsearch configuration without resetting the node?
- To what extent is the new solution improving the current configuration?

Finding the right configuration parameters is difficult when Elasticsearch clusters rely on the amount of data being indexed and the host machine specifications. Moreover, it is also hard to combine different parameters to get the best outcome of configuration parameters.

With the help of machine learning, the above tasks become easier to handle. With machine learning and Elasticsearch, one can utilize the performance and lead to better quality when response time is short, and throughput is high.

1.3 Thesis Outline

This thesis will include the following sections :

- Introduction: Includes the motivation behind this topic and the problem statement.
- Background: Includes literature of Elasticsearch, Parameter Perturbation, and used tools.
- Approach: How the system will look like with details, and what is the approach to implement the solution.
- Implementation: details on the implemented solution
- Results and Discussion: showing the results of the implemented solution. Also, discussing the findings in relation to the problem statement.
- Conclusion and Future work: How the achieved results approve or disapprove of the defined problem. Also, improvements to the delivered solution and how it helps to cover more aspects in the future.

Chapter 2

Background

2.1 Elasticsearch

Elasticsearch is an open-source search engine that is built on top of Apache Lucene [9] using Java. Elasticsearch is used for searching and analyzing purposes. It allows us to search for data using full-text search, analysis, structured search, or different combinations of these three [3]. Elasticsearch provides access to the Application Programmable Interface (API) in an HTTP RESTful API [10], which means less complexity and easier integration with other tools. It also provides near real-time performance [11]. For defining queries, Elasticsearch provides query Domain Specific Language (DSL) that is based on JavaScript Object Notation (JSON). [12]. Elasticsearch is widely used by popular websites such as Wikipedia and Stack Overflow [13]. Elasticsearch has other advantages, such as load balancing and horizontal scalability [14]. The core idea of Elasticsearch is not new at all, as search engines existed before. The difference is that Elasticsearch provides analysis and search of data in real-time with good performance.

2.1.1 Documents

One of the essential terms used in Elasticsearch is documents. A document is an object that includes data which will be stored in Elasticsearch, the documents have the property of being indexed[13]. In other words, when storing data in Elasticsearch, another information can be connected to this stored data. The information describes the stored data and referred to as an index. Indexing helps in enhancing the speed of searching. It is easier to look for a specific type rather than go through all the text word by word. An example of a document is represented in listing 2.1, the document is serialized into JavaScript Object Notation(JSON). The document shown in listing 2.1 represents different keys; those keys are a name, age, "join_date", and accounts. Similarly, those keys have values such as the name "James Smith", age 30, date "2014-06-01", "Instagram" and "Twitter".

```
{
    "name":          "James Smith",
    "age":           30,
    "join_date":     "2014-06-01",
    "accounts": [
        {
            "type": "Instgram",
            "id":   "jamesSm"
        },
        {
            "type": "twitter",
            "id":   "jameySm"
        }
    ]
}
```

Listing 2.1: Example of a Document in JSON

Storing entire documents in the database requires a different way of handling search operations. When dealing with stored data, Elasticsearch does not look on rows of columnar data, rather it filters, searches, and indexes data of stored documents. This makes Elaasticsearch performs well with complex texts.

2.1.2 Clusters

When a running instance or more of Elasticsearch works together to share data, they form a cluster[13]. A cluster contains one or more running instances. The running instance is referred to as a node. In each cluster, there must be an elected master node that holds the responsibilities of deleting or adding nodes as well as adding or removing indexes. The elected master node is not fixed, this means that if a master node breaks down, another node will be elected. Also, using several servers, the nodes will automatically connect to each other if they are on the same network and create cluster [15]. In Figure 2.1 [13] there is a cluster that contains one node as Node1 and this node is marked as the master node.

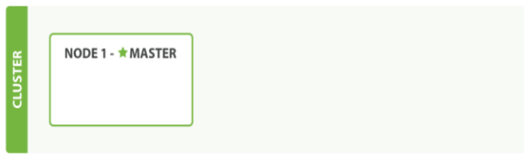

Figure 2.1: Cluster with one node

The cluster in Figure 2.1 has no index. Hence, it has no data. When adding an index to a node, using shards becomes necessary. It is one of the important concepts in Elasticsearch. Shards fix the problem caused by the indexes when storing a large amount of data that requires more of the existing resources of a single node. With shards, the index is subdivided into multiple pieces, and these new pieces are called shards[16][17]. Figure 2.2 [13] shows a cluster with one node as the master node, inside node 1 there are three shards. A shard is considered an independent index. For that, each shard can exist on any node in the cluster.

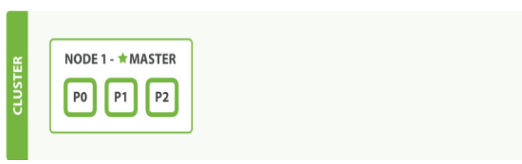

Figure 2.2: A single-node cluster with an index and shards

2.1.3 Elasticsearch Metrics

There are several factors that play a crucial part in the performance of Elasticsearch. However, the metrics are context-dependent, and since different systems can run on top of Elasticsearch [18] more metrics will be considered as well. The following are metrics to consider in Elasticsearch[19];

Cluster Status

Cluster status shows information inside the cluster components, such as running nodes and how many shards are assigned. Also, it provides information on the time it takes a cluster to allocate shards.

Node Performance

The node performance is dependant on the specifications of the machine in which the node is installed. Things like the Central Processor Unit (CPU), memory usage, and Operating System will affect the performance. And since Elasticsearch was built using Java, it is important to investigate the Java Virtual Machine (JVM) metrics as well.

Java Heap

Elasticsearch allocates 32 GB or less to JVM heap of the Random Access Memory (RAM) but never higher. Along with that, Elasticsearch allocates 50 percent of the available RAM or less.

Index Metrics

There are a few parameters that help to optimize and assess index performance. Indexing latency can be calculated by using tools or by using the available parameters **index_total** and **index_time_in_millis**. Another metric is the Flush latency that helps in detecting problems with disks. When there is a problem with slow disks, this flush latency metric will increase.

Search Performance Metrics

Querying is used when using search requests. The number of queries written and how they are written will influence the performance of a node. Because of that, Query Latency and Query Load are two important metrics to monitor.

Both Index and Search performance metrics can be summarized as seen in Figure 2.3, Query Load and Query Latency influence the performance of searching while Index Latency and Flush Latency affect the Indexing Performance.

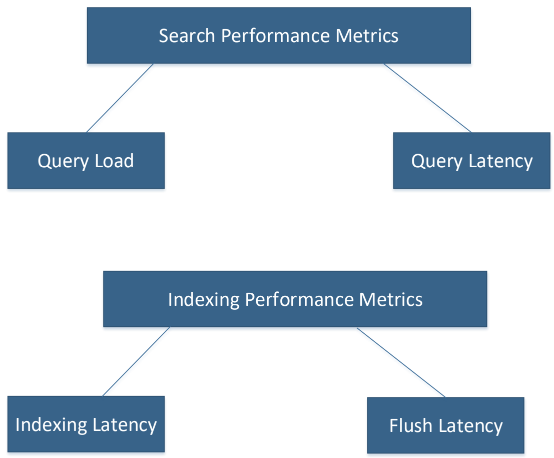

Figure 2.3: Elasticsearch Performance Metrics

Search Requests in Elasticsearch

Having the best performance during search requests is the main goal when using Elasticsearch. When having a large amount of data, the searching will consume more time. In [20], Elsticsearch performance was tested on a cloud environment, the test was to execute six types of queries with different result counts that increasingly vary from query 1 to query 6. In figure 2.4 the X-axes show the six queries while the Y-axis shows the execution time in milliseconds, the page size (PS) is represented in colors. The figure shows that with increasing the page size and result counts, the search time will also increase.

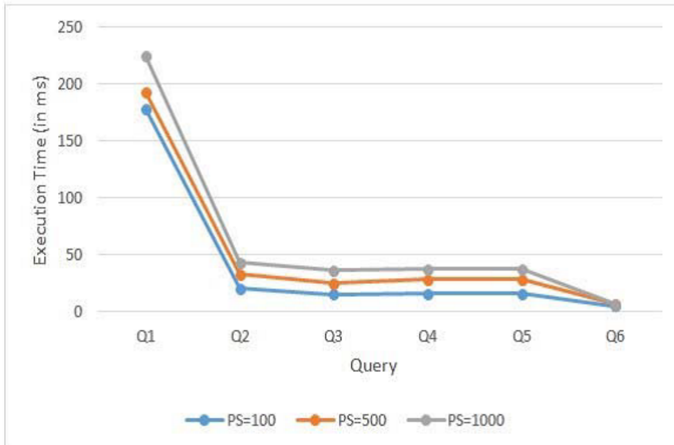

Figure 2.4: Execution Time of Search Queries

Writing a proper search query is the main factor in influencing search performance in Elasticsearch. Similarly, factors like data type and how they are organized play a rule as well. However, to increase the speed of the search there are two methods [19], custom routing, and force merging.

Custom routing

When having several shards in a node, Elasticsearh checks all segments inside each shard, not all shards, only the ones that satisfy the search request. Custom routing gives the ability to store chosen data on the same shard. For that, only one shard will be searched in order to satisfy the query. As a result, it requires less number of shards to investigate rather than going through all shards. Similarly, it is possible to decrease the number of segments of each shard by using Force Merge API [21]

Force Merging

The purpose of Force Merge is to merge segments continuously until the value of **max_num_segments** in a shard is reduced to 1. However, when the number of segments and shards is high, it will become slow to perform the force merging process. For example, merging 10 000 segments to 5000 segments takes less time than merging 10 000 segments to one, this will affect the resources required to perform the process, which will also affect the search requests. In that case, it is recommended to schedule Force Merging on non-busy hours.

2.1.4 Tuning Parameters

There are many parameters to consider when it comes to both searching speed and indexing speed in Elasticsearch. Table 2.1 summarizes the most parameters that have an influence on indexing performance and hence searching performance [22] [23].

Parameter	Description
index.refresh.interval	Time to wait before copying in-buffer memory
index.number.of.replicas	The number of replicas each primary shard has
indices.memory.index.buffer.size	Allocation of heap memory
indices.memory.min.index.buffer.size	Allocation of heap memory
indices.memory.max.index.buffer.size	Allocation of heap memory
index.translog.flush.threshold.size	Make a flush after reaching specific size
index.translog.retention.age	Duration for keeping a translog files
index.translog.sync.interval	How often the translog is synced to disk
index.number.of.shards	The number of primary shards per index
index.shard.check.on.startup	shards should be checked for corruption before opening

Table 2.1: Elasticsearch Tuning Parameters

2.1.5 Logstash

Logstash is an open-source data collector who works on the sever-side with real-time pipe-lining [24]. Although Logstash was mainly created to collect logs, its features are not limited to that, Logstash is capable of handling several use cases. A Logstash instance is capable of receiving an input of different formats, implement some process on the input based on the configuration, and then send the data to several storage systems. Figure 2.5 shows a visualization of Logstash inputs and outputs features. As seen in the figure, Logstash takes different formats as input and provides an output that is capable of being analyzed, archived, monitored, altered, etc. In general, Logstash consists of inputs, filters, and outputs. The input stage will generate an event from the input data format, then at the filter stage, the data will be filtered and will be moved to the output stage in which different tools can be used to manipulate the output as desired[24].

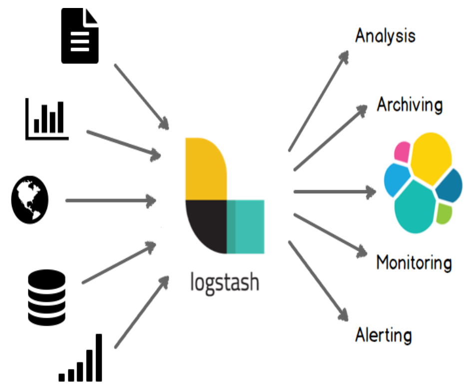

Figure 2.5: Logstash Inputs and Outputs

2.2 ELK Stack

There are many tools when it comes to DevOp field, and those tools serve different purposes from logs to building dashboards. In the same token, combining different tools would help in increasing the efficiency in some aspects within the DevOps field. One of the useful and efficient combinations is the ELK stack [25], which is a collection of three products, Elasticsearch, Logstash, and Kibana, known as ELK stack, it can also include more tools like Filebeat, which is a tool that works to send logs to Logstash. The combination of these three tools is powerful. Starting from Logstash, which will centralize the logging and hence receiving the status of servers in an easy way. On top of Logstash comes Elasticsearch, which will make the process of searching among logs easier. And finally, A graphical interface known as Kibana will provide a useful visualization of logs by building customized dashboards. An example of ELK stack case can be found in the paper [10], in which the ELK stack was used to monitor scientific applications on the Cloud.

One important aspect to mention when discussing ELK stack is the security features. By default, Kibana and Elasticsearch do not provide authentication and authorization for issuing queries [26]. However, this can be enhanced by using third-party plugins such as Search Guard [26] and X-Pack [27], which can secure the ELK stack.

2.3 Elasticsearch Case Studies

Elasticsearch can be implemented within a different context. It is not combined with logging scenarios, it can be applied on top of databases too. This section provides different case studies of Elasticsearch.

2.3.1 NoSQL Databases

The relational database management system is the traditional way of storing data. Oppositely, storing data in a system where a relational database is not used is another approach called NoSQL Databases. NoSQL stands for Not Only SQL , the core idea of NoSQL is also to store unstructured data which can be used when storing document, column databases, etc. [28]. Several NoSQL databases that are commonly used with applications such as MongoDB [29], and Cassandra [30] and many others. Elasticsearch stores documents in the form of JSON, which makes Elasticsearch as NoSQL database since it does not use scripted query language.

Performance on Databases

There are different aspects to investigate when it comes to testing Elsticsearch's performance, and there are several papers that discuss the matter. On a higher level investigation, one can investigate alternative tools than Elasticsearch, such as CouchDB [31] ,which is also NoSQL. Both CouchDB and Elasticsearch perform the primary operations of databases such as insertion, deletion, updating, creation, and selection. However, the two tools have different performance when it comes to time taken to handle the mentioned database operations [32]. In the paper [32], Elasticsearch performed better only on the selection operation while CouchDB showed better results on the rest of the operations. However, when comparing the performance of Elasticsearch versus Relational Database Management System (RDBMS), Elasticsearch performs faster than Relational

Databases such as MySQL [33]. Moreover, the SQL database can be used with Elasticsearch to perform better in searching for data. One example of such a case is presented in [34], where MySQL database and NoSQL were combined to perform faster searching.

2.3.2 Defects Detection

Elasticsearch is capable of serving other purposes that are purely related to software engineering ,such as software testing, as the facts show that maintenance and evolving a system cost over half of the total effort spent on developing the system [35] [36]. By using Elasticsearch, testers can benefit from using Elasticsearch to discover defects faster. In [37], a case study presented how much time it takes to notify bugs by using Time To Notify (TTN) metric and comparing it with other metrics.

2.4 ESRally

ESRally is an open-source tool that helps to benchmark Elasticsearch, and it is available on Github[38]. ESRally provides powerful tasks such as[39]:

1. Executing benchmarks
2. Providing Benchmark data with specifications on the type of data used in the benchmark
3. Helping in finding Elasticsearch performance problems

Also, ESRally supports running as a docker container from the docker image. By providing the Elasticsearch IP when issuing the running command of the ESRally container, it will connect to the existing node on the specified IP and perform the bench-markings.

Besides, the output result will provide a JSON file, which consists of several information as seen in Listing 2.2. The output will include information like the throughput of each operation performed such as indexing and search queries.

```
{
 "rally-version": "1.3.0",
 "environment": "local",
```

```
"trial-id": "1423b4e3-de1b-4a49-b329-692340cad833",
"trial-timestamp": "20190928T104655Z",
"pipeline": "benchmark-only",
"user-tags": {},
"track": "nyc_taxis",
"car": [
 "external"
],
"cluster": {
 "nodes": [
  {
   "node_name": "elasticsearch1",
   "os": {
    "name": "Linux",
    "version": "4.15.0-58-generic"
   },
   "jvm": {
    "vendor": "Oracle Corporation",
    "version": "12.0.1"
   },
   "cpu": {
    "available_processors": 16,
    "allocated_processors": 16
   },
   "memory": {
    "total_bytes": 25253437440
   }
   ],
 "node-count": 1,
 "revision": "de777fa",
 "distribution-version": "7.3.0",
 "distribution-flavor": "default"
},
"results": {
 "op_metrics": [
  {
   "task": "index",
   "operation": "index",
   "throughput": {
    "min": 7919.7283998172215,
    "mean": 7919.7283998172215,
    "median": 7919.7283998172215,
    "max": 7919.7283998172215,
    "unit": "docs/s"
   },
   "latency": {
```

```
        "50_0": 112.03272873535752,
        "100_0": 123.38479235768318,
        "mean": 114.06296049244702
      },
      "service_time": {
        "50_0": 112.03272873535752,
        "100_0": 123.38479235768318,
        "mean": 114.06296049244702
      },
      "error_rate": 0.0
    },
    {
      "task": "default",
      "operation": "default",
      "throughput": {
        "min": 185.11768730001103,
        "mean": 185.11768730001103,
        "median": 185.11768730001103,
        "max": 185.11768730001103,
        "unit": "ops/s"
      },
      "latency": {
        "100_0": 4.116862080991268,
        "mean": 4.116862080991268
      },
      "service_time": {
        "100_0": 4.116862080991268,
        "mean": 4.116862080991268
      },
      "error_rate": 0.0
    }
  ],
  "node_metrics": [],
  "total_time": 15801693,
  "total_time_per_shard": {
    "min": 3,
    "median": 8922,
    "max": 1375645,
    "unit": "ms"
  }
}
```


Listing 2.2: Example of ESRally Output

2.5 Docker

Docker is an open platform tool that is based on the containerization concept[40] and has become a popular tool among companies and developers. Docker came to solve the problem of packages and project dependencies, with docker all of the project code and packages can be built into a docker image and ran by a container that runs an instance of the docker image. As seen in figure 2.6, docker consists of docker daemon, REST API, and Docker Command Line Interface. The combination of these three will allow docker to manage images, containers, data volumes, and docker network.

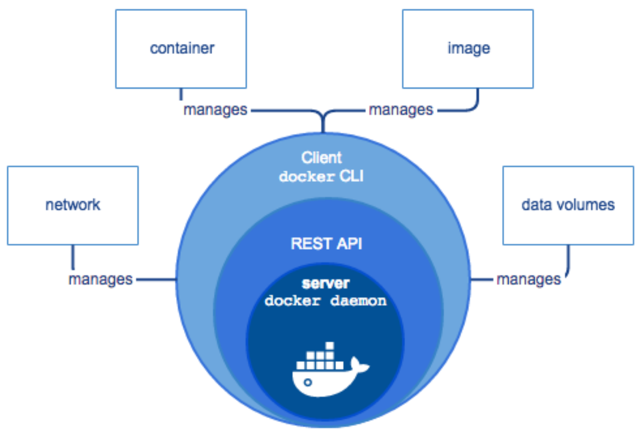

Figure 2.6: Docker Components

2.6 Parameter Perturbation

Tuning Elasticsearch will require an understanding of the parameters used in the configuration and their input values, to achieve this, several parameter values must be entered as an input to a function that returns a perturbed value of the input data. For such a case, stochastic optimization can be used to generate random values from initial ones. Employing such solutions will enable the tuning of Elasticsearch configuration by applying optimization solutions to the parameters. When there is a degree of randomness in input values, stochastic optimization helps in maximizing or minimizing the objective function based on the desired goal. One can design the objective function to maximize throughput or minimize response time ,for example.

2.6.1 Stochastic Approximation

Stochastic Approximation (SA) algorithms are types of solutions for optimization problems. SA helps in solving problems when the objective function has no specific form to analyze but can be approximated based on noisy observations. The noisy observations find an order of parameter estimates which directs the objective function towards zero as in $g(\theta) = 0$ where g is the gradient of the expected objective function as explained in [41] and can be presented as:

$$g(\theta) = \nabla\theta f(\theta)$$

where $\nabla\theta f(\theta)$ is the gradient of the expected objective function

2.6.2 Adaptive Random Search

Adaptive random search is an extension of the random search algorithm, which improves the step size of each iteration in the algorithm [42]. Random search requires a uniform distribution in which it chooses variables from and where those selected variables are independent of other values on other iterations. However, the issue with random search is the step size of each iteration, which makes scaling an issue when the step size is moving within a small range and hence creating a local optima problem. For that, an adaptive random search addresses the step size of the random search. The difference between random search and adaptive random search is that adaptive random search makes a more significant step size to not have a local optima problem as in random search. Both random search and adaptive random search implementation can be found in [43]

2.6.3 Simultaneous Perturbation Stochastic Approximation

Simultaneous Perturbation Stochastic Approximation (SPSA) was introduced in 1992 by Spall [44], which was an improvement of Kiefer-Wolfowitz in [45] and referred to as finite difference stochastic approximation (FDSA). SPSA measures two loss functions that are independent of the number of parameters to optimize. In comparison, FDSA uses one direction at a time, and this means FDSA will perturb only one direction while SPSA will perturb all gradient directions and thus makes it more efficient to use SPSA.

Since the gradient direction of stochastic approximation algorithms may not represent the best direction on an iteration SPSA and FDSA have a disadvantage of slow convergence rate [46]. There have been some proposed papers to help in getting better SPSA implementation [47] [48].

2.7 Related Work

Tuning the configuration to get a better performance has always been a practice among researchers and system admins. It is possible to automate the tuning of systems using machine learning. This has been used in different research papers that use genetic algorithms to reconfigure systems like [49], which handles the tuning of parameters to provide high performance. It uses Apache Drill in a Hadoop cluster, which allows performing different types of querying on top of NoSQL. Also, it automates the reconfiguration of the cluster once the optimal configuration is ready. This thesis is highly related since both serve the purpose of enhancing the performance of clusters. Another thesis that uses the genetic algorithm as a solution [50], this thesis provides a solution for solving high-dimensional problems in Hadoop. The solution consists of using a large population and then evolve them through the cycle of a genetic algorithm. The provided solution, however, does not solve the problem completely as it suggests to combine the solution with other techniques as well. Also, this paper [51] handles the self-tuning of database systems. The paper presents an approach that depends on three inputs to tune its configuration. Those inputs are a number of users, buffer-hit-ration, and size of the database. The approach follows the fuzzy rules, which are defined after some analysis on queries response time.

Similarly, some research papers implement different algorithms to achieve auto reconfiguration tuning. For example, [52] uses the SPSA algorithm to tune the parameters on the Hadoop system. The work in this paper shows the effectiveness of using two system observation per iteration to tune parameters.

Other related work can include self-tuning approaches. For example [53] proposed a self-tuning approach that is based on an artificial neural network, which uses Apache Spark as the system to utilize.

21

Chapter 3

Approach

3.1 Objectives

The objective of this project is to optimize Elasticsearch configuration using an optimization algorithm that is based on Simultaneous Perturbation. The solution will implement the algorithm on a running Elastcisearch container cluster. The goal is to be able to tun the configuration of the cluster by running benchmarks and analyze them through the algorithm and then enhance the configuration on each iteration. For each iteration, Indexing metric and Response Time are the influencing factors on the optimization process.

The benchmark process examines the performance of the running ELasticsearch node by inserting data into it and removing the data once the analyzing is over. However, before running the benchmark, the Elasticsearch node should be up and running with some data. For this reason, the followed approach in this project is to set up the node with data as server logs, product information, and many other types used in real-life scenarios. The solution should provide automatic tuning based on the best of parameters.

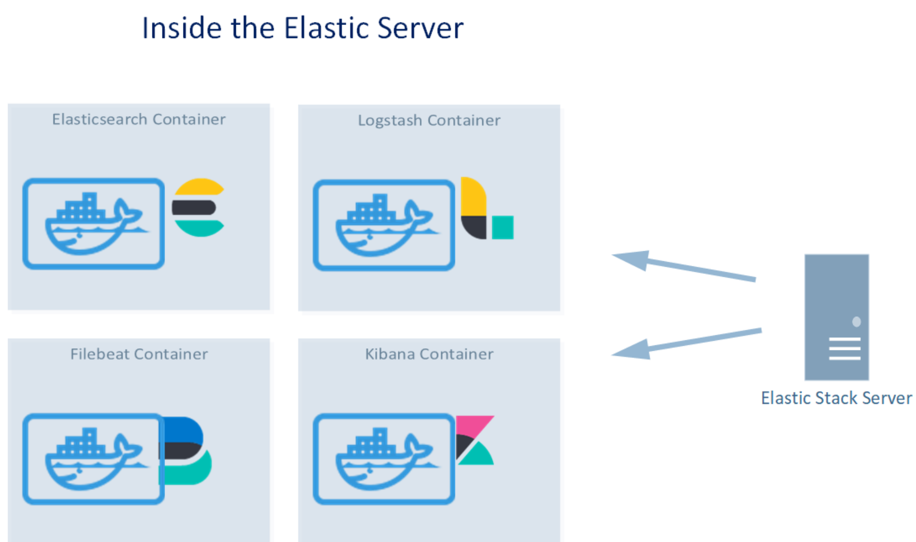

Figure 3.1: Inside Elastic Stack Server

3.2 Infrastructure overview

The infrastructure of the project will consist of one server hosting Elastic Stack applications, and other virtual machines that direct their logs to the Elastic cluster. Also, all applications will be running on docker containers. Figure 3.1 shows the Elastic Stack Server, which will host the Elastic Stack that consists of Elasticsearch, Kibana, Filebeat, and Logstash as docker containers.

In addition to the Elastic server, there will be three virtual machines in which logs will be sent from those virtual machines to the Elastic server. As seen in figure 3.2, the Elastic server will be on a different network than the virtual machines. Figure 3.1 and figure 3.2 will represent the network side of the project since all the docker containers will communicate with each other as well with the other virtual machines.

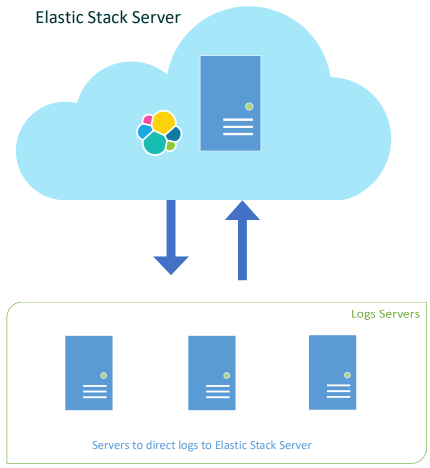

Figure 3.2: Infrastructure Servers

In addition, the docker containers of the Elastic Stack will be using different ports, as seen in figure 3.3 where Elasticsearch uses port 9200, Kibana on port 5601, Logstash on port 5044 and Beats on port 5043.

Continually, the ESRally benchmarking tool is also available as a docker image. Therefore, another container will be used to connect to the existing Elasticsearch node using docker. The algorithm will rely on this combination of docker containers to apply the solution.

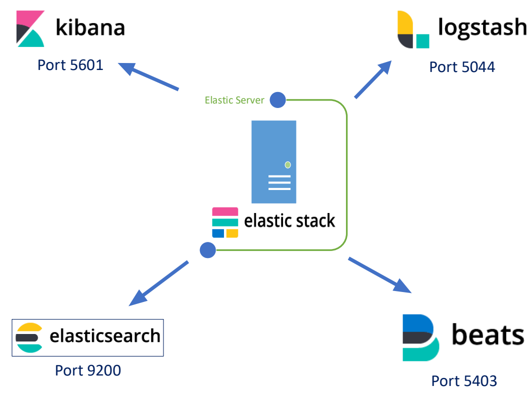

Figure 3.3: Elastic Stack Ports

3.3 Elastic Stack Server Specifications

The main server that holds the docker containers and the algorithm has the following specifications as listed in the table 3.1

Operating System	Linux Ubuntu16.04 xenial
CPU	E5530 @ 2.40GHz
RAM	23 Gb
Disk Space	439G
Hardware Architecture	x86 64
Processor	Intel(R) Xeon(R)

Table 3.1: Elastic Stack Server Specification

This server will be the main machine that will perform most of the tasks in this project.

3.4 Documents Generator

The documents generator is a code written in python that will be connected to the Elasticsearch cluster and generate fake data. The generated data will be in JSON format, which is the format used in Elasticsearch and referred to as documents. Figure 3.4 illustrates the simple steps of the generator, after generating the documents, the code will collect the generated data into bulk and send it to the bulk API of Elasticsearch node. The API request will insert the generated data into the connected Elasticsearch node. For indexing, the code will generate five indexes for five different sets of data.

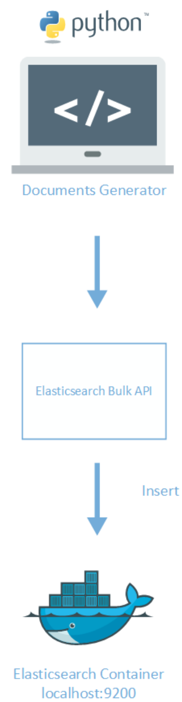

Figure 3.4: Documents Generator Steps

3.5 The Optimizer Algorithm

The optimizer algorithm will implement the concept of SPSA [44] to tune Elasticsearch. This section will breakdown the components of the algorithm that will be used.

3.5.1 Initial Parameters

The number of parameters to tune will be the same during the implementation. However, the initial values of each parameter will be different. For example, *index.referesh.interval* will take the value of seconds like 1s or 100s and so on. While other parameters have a memory size type of value, see table 2.1. In the algorithm, the initial parameters will have random values, and with each parameter, there will be a variable that defines the next step size of that parameter.

3.5.2 Updating Elasticsearch Automatically

Once the initial parameters become defined with their values, the algorithm will update the Elasticsearch cluster using an API from Elasticsearch, which allows updating settings of a node or a cluster. The updating task is simply an HTTP request which will hold information about the parameter that will be updated with their values.

3.5.3 ESRally Benchmarking Tests

As seen in listing 2.2, ESRally provides a JSON file as an output. The output is useful for having an overview of the current performance of the Elasticsearch node. With the statistics included in the JSON file, we can retrieve the mean throughput of indexing and latency of operations.

3.5.4 Elasticsearch Parameters

Elasticsearch configuration includes several parameters. However, the implemented algorithm uses the four parameters listed in table 3.2. A detailed description of the parameters can be found in the official documentation of Elasticsearch for parameter tuning 2.1.

Parameter
translog.sync.interval
indices.recovery.max.bytes.per.sec
index.flush.threshold.size
index.referesh.interval

Table 3.2: The Selected Elasticsearch Parameters

3.5.5 SPSA Algorithm

The goal of SPSA in this thesis is to maximize the objective function of theta $f(\theta)$ where θ is representing the parameters. In this case, θ is a set of Elasticsearch parameters, and Δ represents the step size change that will be added to each parameter on each iteration.

The algorithm implements the following:

1. Let $f(\theta)$ be the system performance when the set of parameter equals θ.
2. Let Δ_n be the step size change that will be added to each parameter where n represents the current number of the iteration.
3. Let $\Delta_{n_{SZ}}$ be the minimum step size of each parameter, and n represents the current number of the iteration.

$$f(\theta_n) = (\frac{f(\theta_{n-1}+\Delta_n) - f(\theta_{n-1}-\Delta_n)}{\left(\frac{f(\theta_{n-1}+\Delta_n) + (\theta_{n-1}-\Delta_n)}{2}\right)} * 100 + 1) * \Delta_{n_{SZmin}}$$

The implemented algorithm in this project is presented in Algorithm 1. The algorithm starts with initial parameters θ, then for each iteration, the algorithm will generate a perturbation vector Δ.

Algorithm 1 Simultaneous Perturbation Stochastic Approximation

Initial parameters $\theta \in \mathbb{R}$
Initial Step Size for each parameter Δ
for $n = 1, 2, \ldots, N$ **do**
  Generate perturbation vector $\Delta_n \in \mathbb{R}$
  Compute $f(\theta) = \Delta_n + \theta_{n-1}$
  Compute $f(\theta) = \Delta_n - \theta_{n-1}$
  Calculate percentage difference of $f(\Delta_n + \theta_n)$ and $f(\Delta_n - \theta_n)$
  Calculate new step size Δn for each parameter
  Update θ from the best of $f(\Delta_n + \theta_n)$ and $f(\Delta_n - \theta_n)$
end for

3.5.6 Objective Function

The objective function in the algorithm will use ESRally output to get the indexing throughput and latency time of other operations. The objective function formula :

$$f(x) = \frac{Indexing}{ResponseTime}$$

where *indexing* is the mean indexing throughput, and that is the number of indexed documents, and *Response Time* is the mean latency time of operations, and that is the latency of different search queries types being performed on the Elasticsearch cluster.

3.5.7 Step Size

The step size is a value assigned to each parameter, and this value will decide what the size of the next step is. For each parameter, there is a minimum and maximum value, the step size will also have a minimum and a maximum value that is based on the parameter value range. So for Pn where Pn is a parameter:

$$PnSZmin = \frac{Pn_{max} - Pn_{min}}{100}$$

where $PnSZmin$ is the minimum step size of parameter n , and Pn_{min} is the minimum value of parameter n, and Pn_{max} is the maximum value of parameter n. And the maximum value of the step size for Pn is presented as follows:

$$PnSZmax = \frac{Pn_{max} - Pn_{min}}{10}$$

where $PnSZmax$ is the maximum step size of parameter n , and Pn_{min} is the minimum value of parameter n, and Pn_{max} is the maximum value of parameter n. Now for each iteration, the step size value will be updated for each parameter depending on the objective function of each $f(+\theta)$ and $f(-\theta)$. The new step size value will be calculated following the below formula:

$$StepSize = PnSZmin * (1 + DP)$$

where *DP* is the difference percentage of $f(+\theta)$ and $f(-\theta)$ and *DP* $\in$ *[0,1]*. If step size value exceeds the maximum value of the step size, then the new step size value will be the maximum step size $PnSZmax$.

3.5.8 Optimizer Data Flow

The optimizing algorithm can be presented in a data flow diagram, as in figure 3.5. The data flow diagram starts with *Initial Set of Parameters with Step Size Values*, and that is, the parameters in which will be tuned in the algorithm. For each parameter, there is a step size that will be updated each iteration. Then *Calculate the negative and positive paths of each variable*, where $+\theta$ presents a set of values to add to each parameter from the selected set of parameters, and $-\theta$ is the negative values of these values. After that, *Update Elasticsearch settings* with both $+\theta$ and $-\theta$, then *Calculate objective function of* $+\theta$ *and* $-\theta$ *and percentage difference*, if the iteration is not the last one then check which objective function value is better is it $+\theta$ or $-\theta$, the best value will be then updated either the stage *Update set of parameters as in* $+\theta$ *with new step size value* or the $-\theta$ one as seen in data flow diagram 3.5. Then the new set of parameters will be used to update the Elasticsearch settings as before. If the iterations are over, then update Elasticsearch with the best set parameters from the previous iterations.

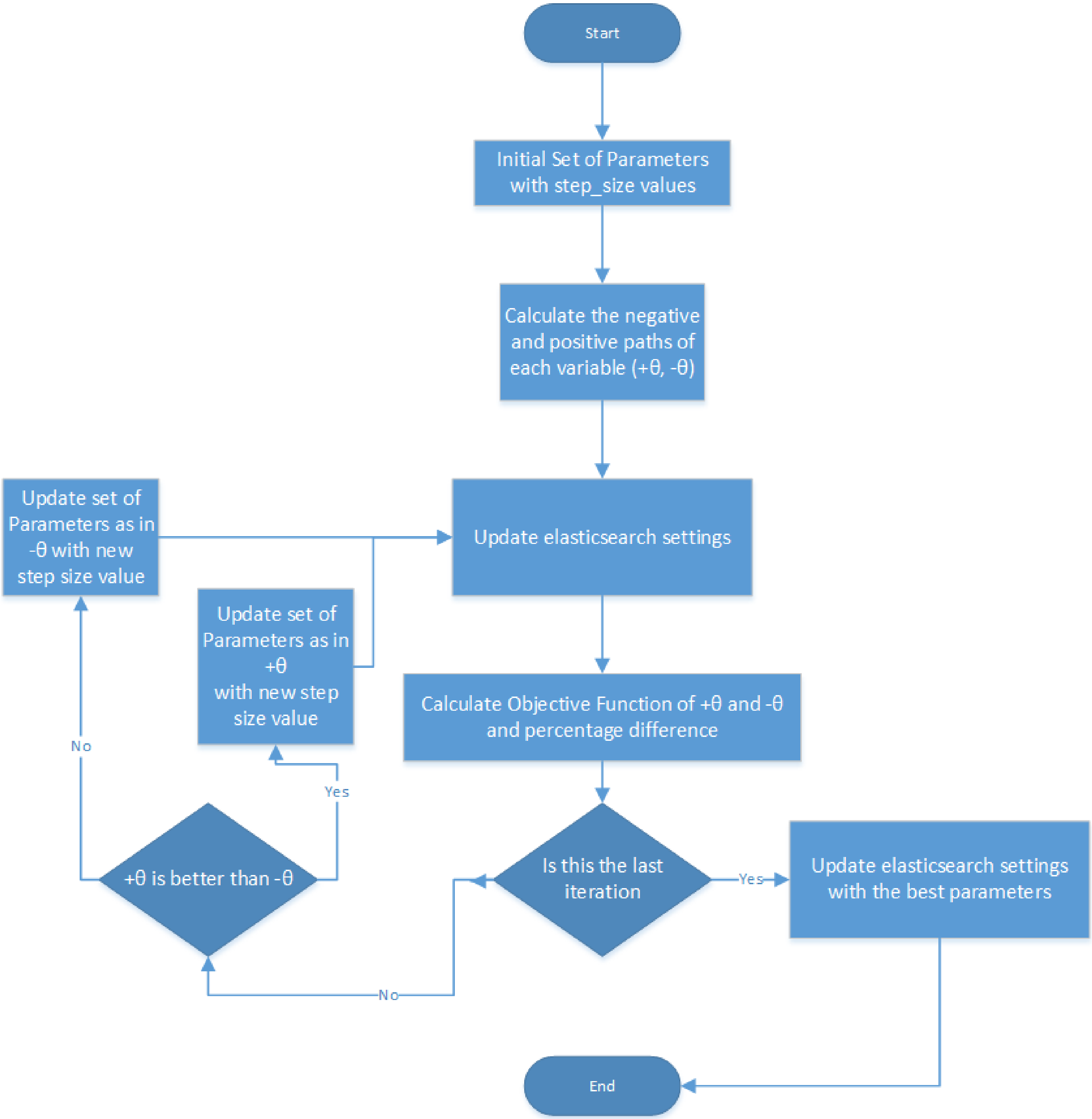

Figure 3.5: Optimizer Algorithm Data Flow

Chapter 4

Implementation

4.1 Docker compose

Docker containers can be defined in a docker-compose file that will hold the container information like name, ports, environment variables, and so on. For the application used in this project, the elastic stack was divided into four containers; Elasticsearch, Logstash, Kibana, and Filebeat. See full file in appendix A

A short definition of the Elasticsearch container includes the docker image, container name, and some environment variable definition like node name and cluster name, see code 4.1

```
compose Elasticsearch ,label=compose−elastic]
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.3.0
    container_name: elasticsearch1
    environment:
      − node.name=elasticsearch1
      − cluster.name=docker−cluster
```

Listing 4.1: Elasticsearch Docker Compose

Similarly, the definition of Logstash container will include a build, and that is referring to the Dockerfile in the same directory and building it as a Docker image. Also, since Logstash will use port 5000, it is essential to define it in the container. See code 4.2

```
logstash:
  container_name: logstash
  build:
    context: .
    dockerfile: Dockerfile-logstash
  restart: always
  ports:
   - "5000:5000"
```

Listing 4.2: Logstash Docker Compose

For Kibana, which is the tool that visualizes Elasticsearch, the container definition of it will include the docker image name, the container name, and, most importantly, the Elasticsearch IP to connect to which has a node running. See code 4.3

```
kibana:
  image: docker.elastic.co/kibana/kibana:7.3.0
  container_name: kibana
  environment:
    ELASTICSEARCH_HOSTS: http://localhost:9200/
```

Listing 4.3: Kibana Docker Compose

4.2 Data Generator

The data generator is a python script that uses a library that generates different types of data and then converts them into a JSON format. It also uses the Elasticsearch client for Python, this makes it easier to connect to an existing cluster and to use the Elasticsearch APIs like the Bulking API. For the full Python script of the data generator see appendix B. To break it down, the script creates the following types of data; person, hardware, Internet, File, Unit system, Address, and food data. It starts with generating data of type persons. The data includes the fields first name, last name, academic degree, and email. The implementation method in Python is like Listing 4.4. The fields will hold random values every time the method gets called by the code.

```

```

```
def person_data(self,length):
    for i in range(length):
        yield    {
        "_index": "person."+date_today,
        "doc": {
            "last_name": person.surname(),
            "first_name": person.name(),
            "Academic_degree": person.academic_degree(),
            "email": person.email()
        }
    }
```

Listing 4.4: Person Data Generator

The hardware data includes different fields of different hardware information like CPU, RAM size, RAM type, type of memory SSD or HDD, Manufacturer, etc. The Python method for the hardware data generator presented in listing 4.5

```

    def hardware_data(self,length):
        for i in range(length):
            yield    {
            #"_id": str(uuid.uuid4()),
            "_index": "hardware."+date_today,
            "doc": {
                "cpu": hardware.cpu(),
                "frequency": hardware.cpu_frequency(),
                "codename": hardware.cpu_codename(),
                "model_code":hardware.cpu_model_code(),
                "generation":hardware.generation(),
                "manufacturer":hardware.manufacturer(),
                "graphics":hardware.graphics(),
                "phone_model":hardware.phone_model(),
                "ram_size": hardware.ram_size(),
                "ram_type": hardware.ram_type(),
                "resolution": hardware.resolution(),
                "ssd_or_hdd": hardware.ssd_or_hdd(),
                "screen_size": hardware.screen_size()
            }
        }
```

Listing 4.5: Hardware Data Generator

For address data, a general address will be generated, which includes information like country code, country, city, current local, latitude, and longitude. Listing

4.6 presents the Python method that generates address data as JSON.

```

    def address_data(self,length):
        for i in range(length):
            yield     {
            #"_id": str(uuid.uuid4()),
            "_index": "address."+date_today,
            "doc": {
                "address": address.address(),
                "country_code": address.country_code(),
                "country": address.country(),
                "continent":address.continent(),
                "city":address.city(),
                "current_locale":address.get_current_locale(),
                "coordinates":address.coordinates(),
                "latitude":address.latitude(),
                "longitude": address.longitude()
            }
        }
```

Listing 4.6: Address Data Generator

The internet data includes information like IP and IPv6. It also provides information about home pages and HTTP requests. Listing 4.7 presents the Python method to generate Internet data.

```
    def internet_data(self,length):
        for i in range(length):
            yield     {
            "_index": "internet."+date_today,
            "doc": {
                "content_type": internet.content_type(),
                "ip": internet.ip_v4(),
                "ip_v6": internet.ip_v6(),
                "emoji":internet.emoji(),
                "home_page":internet.home_page(),
                "network_protocl":internet.network_protocol(),
                "mac_address":internet.mac_address(),
                "user_agent":internet.user_agent(),
                "port": internet.port(),
                "http_method": internet.http_method(),
                "http_status_code": internet.http_status_code(),
                "http_status_message": internet.http_status_message()
            }
```

```
        }
```

Listing 4.7: Internet Data Generator

Unit system data includes only two fields, unit, and prefix. See listing 4.8 for the Python method.

```
def unit_system_data(self,length):
    for i in range(length):
        yield    {
        #"_id": str(uuid.uuid4()),
        "_index": "unit_system."+date_today,
        "doc": {
            "unit": unit_system.unit(),
            "prefix": unit_system.prefix(),
        }
    }
```

Listing 4.8: Unit System Data Generator

Food data includes information like vegetables, fruit, dish, drink, and spices. Refer to listing 4.9 for the Python code to generate food data.

```
def food_data(self,length):
    for i in range(length):
        yield    {
        #"_id": str(uuid.uuid4()),
        "_index": "food."+date_today,
        "doc": {
            "vegetable": food.vegetable(),
            "fruit": food.fruit(),
            "dish": food.dish(),
            "drink":food.drink(),
            "spices":food.spices(),
            "current_locale":food.get_current_locale(),
        }
    }
```

Listing 4.9: Food Data Generator

Finally, all those methods will generate data in JSON format, which will be used in the bulk API of Elasticsearch to index them into the cluster.

4.3 Running ESRally

ESRally can be run using Docker container, the command to run ESRally is:

```
$docker run elastic/rally --track=nyc_taxis --test-mode --
  pipeline=benchmark-only --target-hosts=elasticsearch:9200
```

Where –track is the designed queries and indices to test and insert in the Elasticsearch. And –pipeline is a number of steps that are performed to get results from the benchmark. And finally, the –target-hosts is the Elasticsearch node that will be tested.

4.4 SPSA Algorithm

The SPSA implementation in Elasticsearch includes several Python files :

- ESRallyConnector: Responsible for the connection with the ESRally tool. See appendix C
- Parameters: Elasticsearch parameter definitions will go through this file. See appendix D
- Perturbation Optimizer: Perform the actual algorithm. See appendix E
- Race Reader: This file will read the JSON output from ESRally, it converts the output of JSON to variables in the code to calculate the objective function later. See appendix F

```

def SPSA(self,current_par, max_iter):
        # Defining variables
        # current_par will include the parameters values
        # max_iter will decide how many iterations to perform

        best_of_best = {}
        best = {}
        candidate = {}
        self.x_plus_minus_improvement = 0
        self.current_par = current_par

        #the iterations will happen in this loop
        for i in range(max_iter):
```

```
        X_positive, X_negative = self.x_plus_minus(self.current_par)
        candidate['vector_positive'] = X_positive
        candidate['vector_negative'] = X_negative


        # get objective function of the X+ and X-
        of_plus = self.takes_vector(candidate['vector_positive'])[0]
        of_minus = self.takes_vector(candidate['vector_negative'])[0]

        # get the improvement percentage of of+ and of-
        self.x_plus_minus_improvement = self.calculate_percentage_difference(float(of_plus), float(of_minus))

        # find which of (x+ x-) is best
        if (of_plus > of_minus):
            best['cost'] = of_plus    best['vector']=candidate['vector_positive']
            #update the new parameter's values from the X+ parameter values
            self.update_parameters(best)

        else:
            best['cost'] = of_minus best['vector']=candidate['vector_negative']
            self.update_parameters(best)

        #This will save the best value in all iterations
        if not best_of_best or best['cost'] > best_of_best['cost']:
            best_of_best['cost'] = best.get('cost')
            best_of_best['vector'] = best.get('vector')

        #update the step size using this method
        self.update_step_size()

    return best_of_best
```

Chapter 5

Result and Discussion

5.1 Test Designs

This section handles the test's design and specifications that were performed. For each test, there is an indexing operation, and that is, inserting new data to the Elasticsearch cluster to test the performance during the indexing task. Then, there are several types of operations that will be performed on those indexed data, and that is, several types of search queries. Besides, each test will have a list of specifications like the following list:

1. The number of documents: The number of data inserted in Elasticsearch, a document is represented by a JSON object.

2. Bulk Size: How many documents to index per request

3. Type of Operations: Performing different types of search queries like aggregations, range, match all, etc.

4. The number of iteration per operation: Keep repeating an operation for a specific number.

5.1.1 Taxi rides data

The Taxi Rides documents will include data like pickup location, pickup dateline, passenger account, improvement surcharge, etc. See listing 5.1


```
{
    "total_amount": 28.3,
    "improvement_surcharge": 0.3,
    "pickup_location": [-73.9931869506836, 40.66499328613281],
    "pickup_datetime": "2015-01-01 00:39:28",
    "trip_type": "1",
    "dropoff_datetime": "2015-01-01 01:17:07",
    "rate_code_id": "1",
    "tolls_amount": 0.0,
    "dropoff_location": [-73.91593933105469, 40.7042236328125],
    "passenger_count": 1,
    "fare_amount": 27.0,
    "extra": 0.5,
    "trip_distance": 5.72,
    "tip_amount": 0.0,
    "store_and_fwd_flag": "N",
    "payment_type": "2",
    "mta_tax": 0.5,
    "vendor_id": "2"
}
```


Listing 5.1: Example Taxi Ride Document

Test Specifications

1. Number of documents: 16 5346692
2. Bulk Size: 10 000
3. Type of Operations: 6
4. Number of iteration per operation: 100

5.1.2 Geo-Names Data

Geo-names data represents geographical information about specific areas, as seen in listing 5.2, such data includes country code, population, timezone, etc.

```json
{
"geonameid": 3039805,
"name": "Montaup",
"asciiname": "Montaup",
"feature_class": "L",
"feature_code": "AREA",
"country_code": "AD",
"admin1_code": "02",
"population": 0,
"dem": "2243",
"timezone": "Europe/Andorra",
"location": [1.58156, 42.58328]
}
```

Listing 5.2: Example Geonames Docuemnt

Test Specifications

1. Number of documents: 11 396505
2. Bulk Size: 5000
3. Type of Operations: 22
4. Number of iteration per operation: 100

5.1.3 HTTP Log Data

Http logs data represents Http requests. Htpp requests include information like the type of request, the status of the request, size of the request, and the client IP. See listing 5.3.

```json
{
  "@timestamp": 898459201,
  "clientip": "211.11.9.0",
  "request": "GET /english/index.html HTTP/1.0",
  "status": 304,
  "size": 0
}
```

Listing 5.3: Example HTTP Log Docuemnt

Test Specifications

1. Number of documents: 27 08746
2. Bulk Size: 5000
3. Type of Operations: 10
4. Number of iteration per operation: 100

5.2 Objective Function Results

This section will show the results of the objective function in all iterations for several types of data. For each iteration, there are two objective function results, the direction of the tuning will follow the one with the better result.

5.2.1 Taxi rides data

Figure 5.1 shows the tuning performance of the algorithm on Taxi Rides data. At the first iteration, the objective function value was 19547.21, while the value was 21381.25 at the last iteration. On iteration 2, the value increased to 24040.57, and since the goal was to maximize the objective function, the higher the value, the better. However, on iteration 3, the value decreased to 12525.87 as the lowest drop down of this test. This means that the system at iteration 3 did not perform well within the testing context. However, the value of the objective function kept increasing and decreasing from an iteration to another, on iteration 78 the objective function reached 26619.54 as the largest value among other iterations. This means that the set of parameters used at iteration 78 gave the best performance.

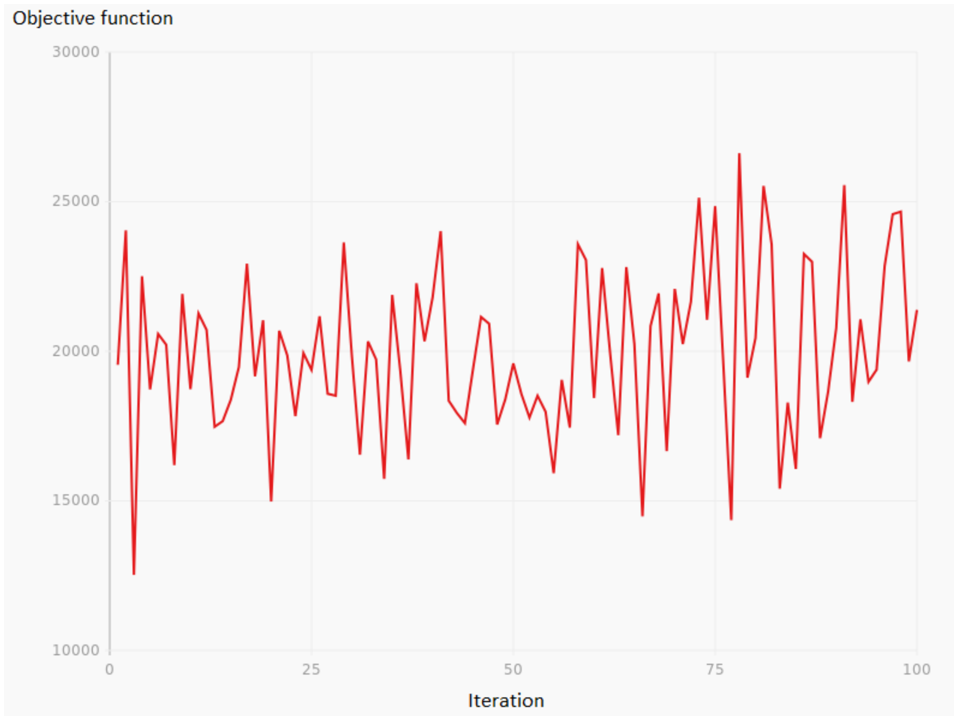

Figure 5.1: Tuning Iterations - Taxi Rides Data

Figure 5.2 is similar to figure 5.1, however the difference is that on each iteration there are two results of the objective function, which are the results of both directions, the dots with line represent the best direction and hence the direction for the tuning process, while the dots without a line are the ones with the worse tuning direction. For example, at iteration 78, which the tuning process was at its best with a value of 26619.54, the other tuning direction was 19985.26. However, SPSA will choose the best path of these two. In that case, the system tuning process followed the path with a value of 26619.54.

Figure 5.2: Directions of Objective Function - Taxi Rides Data

5.2.2 Geo-Names Data

Figure 5.3 shows the tuning performance of the algorithm on Geo-names data. The starting value of the objective function was 46799.96, while it ended with a value of 47432. The lowest performance was at iteration 33 with the value of 33920.05. Oppositely, at iterations 31, 68, and 49, the values of the objective function reached 62659.34, 63690.46, and 61740.11 while the most significant value reached 67200.09 at iteration 17. This indicates that the set of parameter values at iteration 17 showed the best performance of the system during the test.

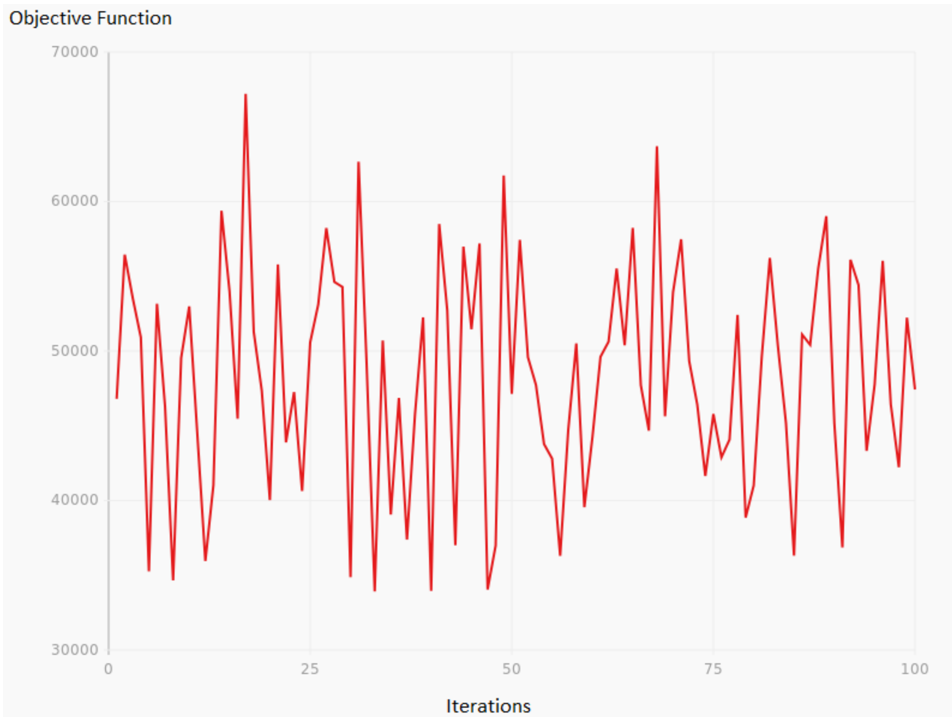

Figure 5.3: Tuning Iterations - Geo-names

Figure 5.4 is similar to figure 5.3. The difference is that figure 5.3 presents all objective function results on each iteration while figure 5.4 only presents the tuning process direction that was chosen by the algorithm. For example, at iteration 17, which was the largest objective function result with a value of 67200.09, the opposite direction for that iteration was valued 49285.52, the algorithm at iteration 17 decided to go for the better value to achieve better performance. Figure 5.4 makes it easy to observe the performance of the algorithm by comparing the dots with the lined dots.

Figure 5.4: Tuning Iterations - Geo-names Data

5.2.3 HTTP Log Data

Figure 5.5 shows the tuning performance of the algorithm on HTTP logs. On the first iteration, the objective function value was 11925.94 at the first iteration. It then increased to 12357.94 at iteration 4, which was the second-largest value of this test. The tuning process reached its best at iteration 42 with the value of 13996.42. Also, the lowest value was at iteration 67, with a value of 7126.78. And finally, at the last iteration, the objective function value was 10300.35. This means that the set of parameter values of iteration 42 are the best set for performance in comparison with the other iterations.

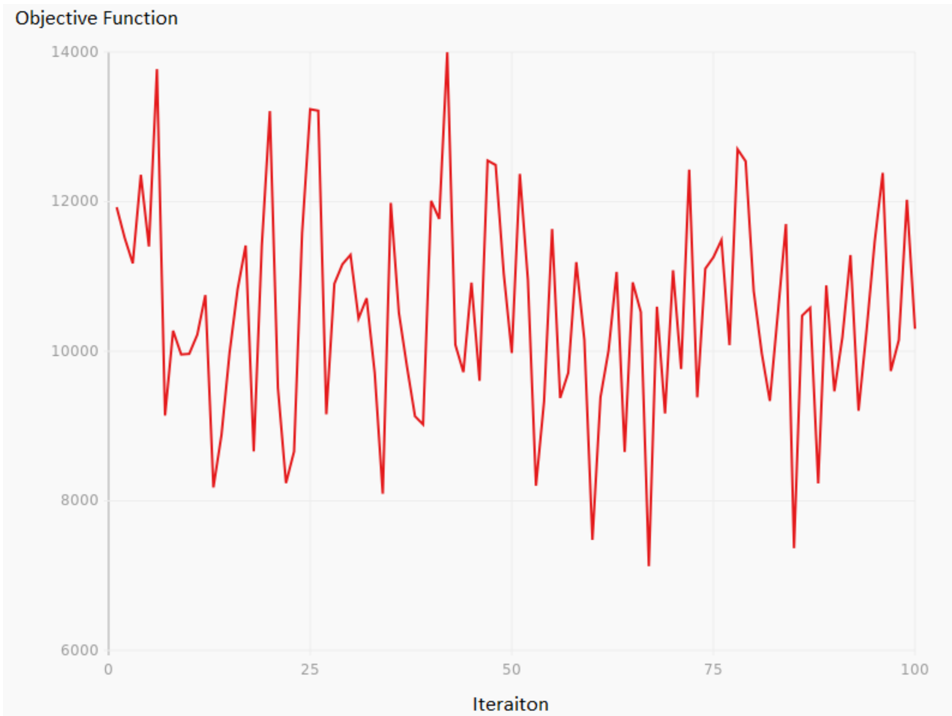

Figure 5.5: Tuning Process - HTTP Logs Data

Figure 5.5 showed the tuning process of the system. However, the algorithm performs two tests in each iteration and chooses the best direction of the two. Figure 5.6 shows all results on each iteration, and the dots with a line are the ones with the best path while the dots without a line are the ones with the worse path. For example, the largest value, hence the best performance, was at iteration 42 with a value of 13996.42. On that same iteration, the opposite direction was 11129.87.

Figure 5.6: Directions of Objective Function - HTTP Logs Data

5.3 Number of Indexed Documents

This section will show the results of several documents indexed in Elasticsearch node. Each test includes 200 Elasticsearch indexing operations, that is, on each operation, some documents are inserted in Elasticsearch node and indexed. From the same test results of the objective function, the indexing operation performance will be shown in this section.

5.3.1 Taxi Rides Data

Figure 5.7 shows the process of the indexing operation on Taxi Rides data. The operation was performed 200 times. The Y-axes presents the number of documents indexed per second, and the X-axes presents the iteration number of the indexing operation. The starting performance showed that 7595 documents were indexed per second. It then dropped down to 7025 documents/second. Similarly, the lowest drop down was at indexing operation number 186 in which the number of documents indexed was 4318.19 document/second. However, the best performance was at iteration 193 with 9390 documents/second.

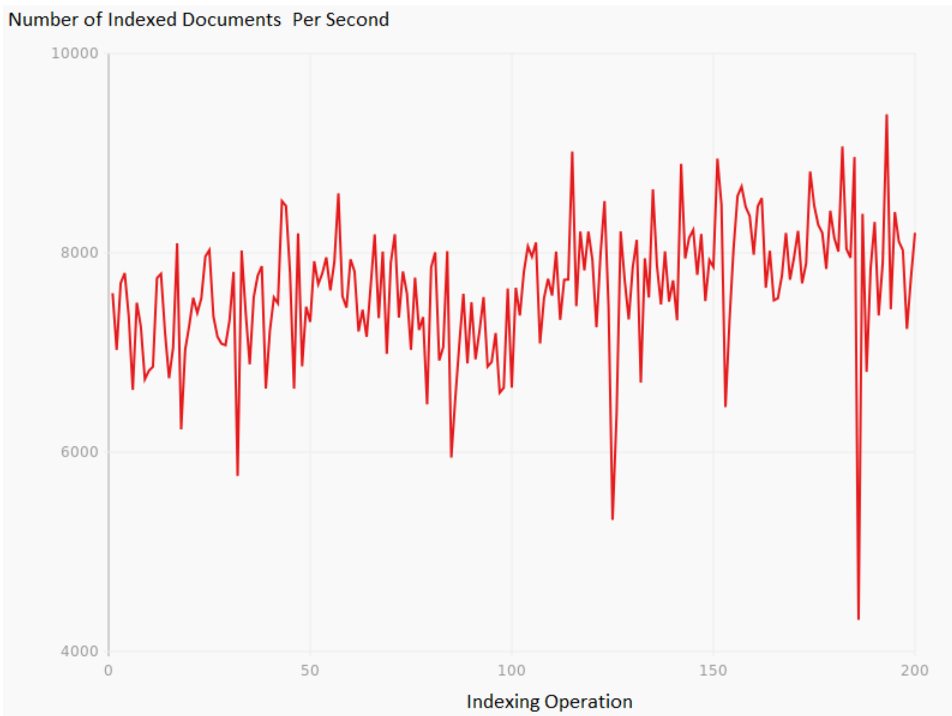

Figure 5.7: Number of Documents Indexed - Taxi Rides Data

5.3.2 Geo-Names Data

Figure 5.8 shows the process of the indexing operation on Geo-names data. The operation was performed 200 times. The starting performance showed that 4001 documents were indexed per second. It then increased at its best to reach 5594 documents/second. However, the lowest drop down was at indexing operation number 77 in which the number of indexed documents was 2996.43 document/second.

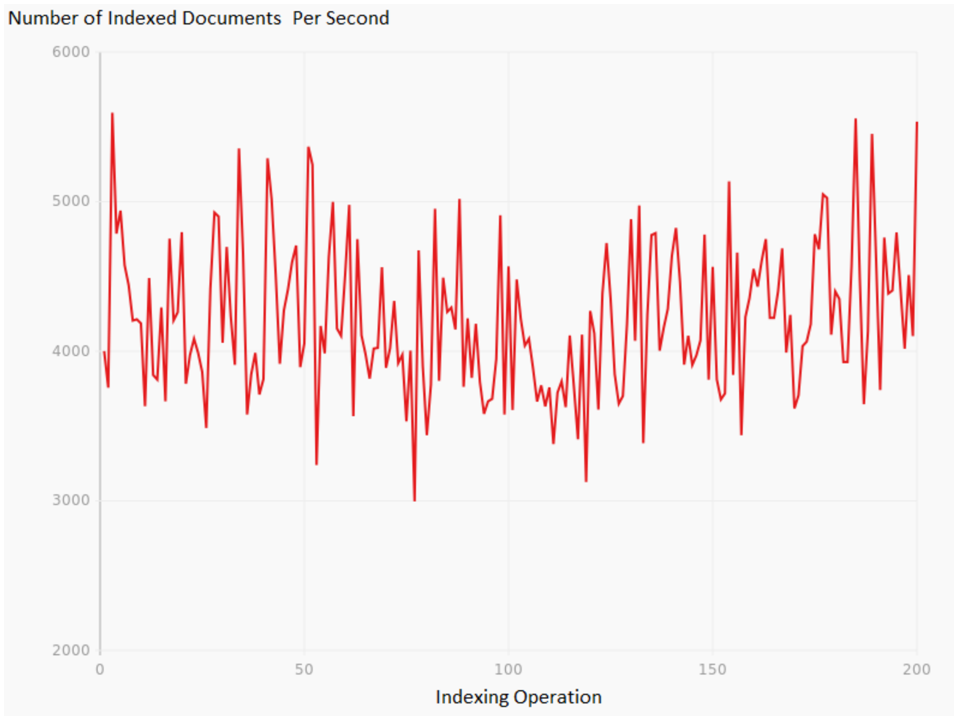

Figure 5.8: Number of Documents Indexed - Geo-names Data

5.3.3 HTTP Log Data

Figure 5.9 shows the process of the indexing operation on HTTP logs data. The starting performance showed that 4339 documents were indexed per second. But then it dropped to 1356 documents/second at index operation number 3 as the lowest result. However, the top performance was at iteration 52 where the number of indexed documents was 4998 documents/second.

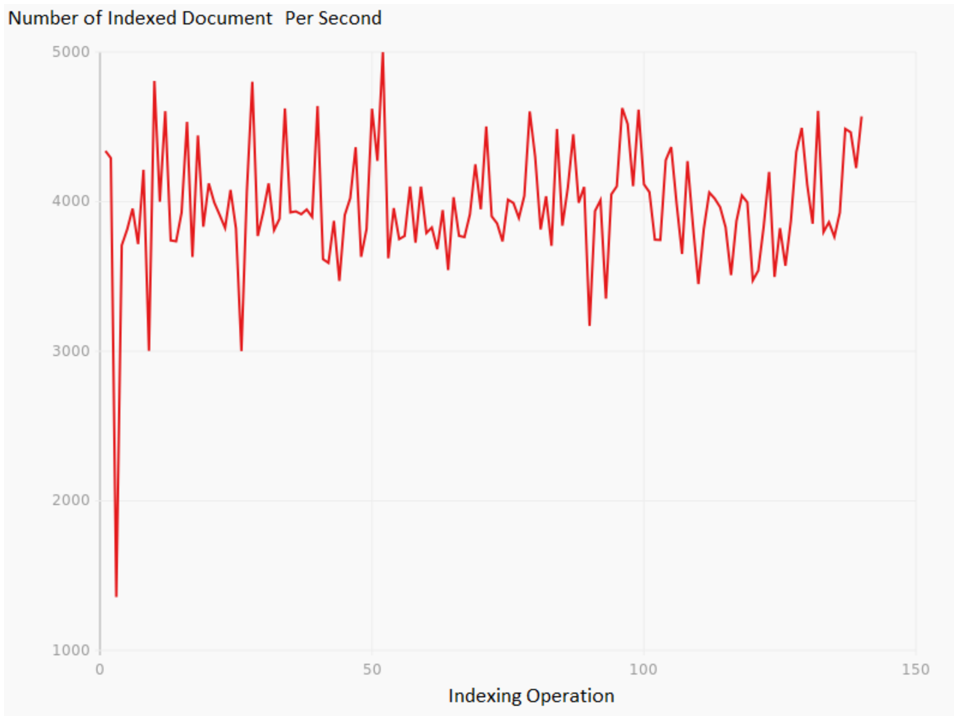

Figure 5.9: Number of Indexed Documents - HTTP Logs Data

5.4 Discussion

This section discusses the different challenges in this project. It also evaluates the work done in this report. The goal of this thesis was to write an SPSA algorithm that would give a solution to tune the performance of Elasticsearch.

5.4.1 Problem Statement

This thesis solves the following problem statement: *How to achieve better Elasticsearch performance by applying Simultaneous Perturbation Stochastic Approximation algorithms while Elasticsearch cluster keeps on scaling.*

However, to solve the problem statement, one has to break down the problem statement into a few questions:

- How to dynamically change Elasticsearch configuration without resetting the node?
- To what extent the new solution is improving the current configuration?

The goal of this thesis was to use optimization solutions to tune Elasticsearch configuration to get a better performance. The implemented algorithm for the optimization was SPSA. The SPSA algorithm iterates several times by testing the current performance of the system. On each iteration, SPSA will observe the system by applying a set of parameters, which is a representation of a parameter combination values. Those values are updated every iteration, and the updates are dependant on the previous iteration in which the system performance was observed.

Finding the correct set of parameters that would tune the performance is difficult when the Elasticsearch node is storing more data and performing more queries. Therefore, using optimization solutions such as SPSA improves the process of choosing the set of parameters that would provide good performance.

The implemented SPSA algorithm provides dynamic tuning to the Elasticsearch nodes. It keeps performing tests on the node while observing the performance. The best solution will be saved, and once all tests are done, the Elasticsearch node will be updated with the best solution without any need to reset the node.

Configuring Elasticsearch has to be done manually by providing which parameters to change to reach a better performance. This makes it impractical when continuous data is being inserted and indexed with respect to the hosting machine specifications. Therefore, the proposed solution provides automatic reconfiguration by testing a different set of parameters and updating the configuration with the best solution.

Chapter 6

Conclusion And Future Work

6.1 Future Work

Although the results achieved in this work were satisfactory, there is more to build on top of it. This chapter gives suggestions to carry on with the current work.

6.1.1 Improvements

This work can be improved by performing more tests with more parameters. Moreover, in Elasticsearch, there are static parameters, and those parameters cannot be updated without resetting the node. Therefore, covering the static parameters would improve the work.

6.1.2 Features

Since Elasticsearch can be used for different purposes, a good feature would allow the prioritization of indexing throughput over latency and vice versa. This will help scenarios that are totally focusing on indexing operations or response time.

In addition, it is possible to add different optimization algorithms to the tool. By following the same tuning process, increasing the optimization methods would be sufficient.

6.1.3 Framework

The scope of this thesis covers a few parameters with the tuning algorithm. However, this can be further enhanced towards building a framework in which all Elasticsearch parameters are defined. With such a framework, it would be possible to define which parameters to tune on the run time. Also, adding several tuning algorithms to the framework would give the option to the system admins to test based on their desires.

6.2 Conclusion

The main goal of this thesis was to enhance the performance of Elasticsearch by implementing an optimization algorithm to tune its parameter configuration. After several tests, the Elasticsearch node showed improvements in its performance after implementing the simultaneous perturbation stochastic approximation algorithm. The algorithm implemented in this thesis relies on observing the system two times on each iteration, and based on that, the tuning process takes a step toward optimizing the configuration.

Being able to tune the configuration using different types of data means that the algorithm can find a better set of values for specific kinds of data. Hence, finding an optimized version of the configuration where both latency and indexing are quality factors.

Performing several experiments on the designed system resulted in efficiently tuning the Elasticsearh parameters. Overall, the Elasticsearch configuration and the implemented algorithm relied on the latency of operations and the number of inserted documents. As a result, the configuration adapted to a better set of values, which serves the objectives of this report.

Bibliography

[1] (2018). Data never sleeps 6.0, www.Domo.com, [Online]. Available: https://www.domo.com/assets/downloads/18_domo_data-never-sleeps-6+verticals.pdf (visited on 05/10/2019).

[2] A. Jacobs, 'The pathologies of big data', *Communications of the ACM*, vol. 52, no. 8, pp. 36–44, 2009.

[3] (2019). Elasticsearch, [Online]. Available: https://www.elastic.co/ (visited on 08/10/2019).

[4] (2019). Stories from users like you, www.elastic.co, [Online]. Available: https://www.elastic.co/customers/ (visited on 05/10/2019).

[5] L. König and A. Steffens, 'Towards a quality model for devops', *Continuous Software Engineering & Full-scale Software Engineering*, p. 37, 2018.

[6] C. Ebert, G. Gallardo, J. Hernantes and N. Serrano, 'Devops', *Ieee Software*, vol. 33, no. 3, pp. 94–100, 2016.

[7] C.-P. Bezemer, S. Eismann, V. Ferme, J. Grohmann, R. Heinrich, P. Jamshidi, W. Shang, A. van Hoorn, M. Villaviencio, J. Walter *et al.*, 'How is performance addressed in devops? a survey on industrial practices', *arXiv preprint arXiv:1808.06915*, 2018.

[8] A. F. Nogueira, J. C. Ribeiro, M. Zenha-Rela and A. Craske, 'Improving la redoute's ci/cd pipeline and devops processes by applying machine learning techniques', in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, IEEE, 2018, pp. 282–286.

[9] M. McCandless, E. Hatcher and O. Gospodnetic, *Lucene in action: covers Apache Lucene 3.0*. Manning Publications Co., 2010.

[10] S. Bagnasco, D. Berzano, A. Guarise, S. Lusso, M. Masera and S. Vallero, ‘Monitoring of iaas and scientific applications on the cloud using the elasticsearch ecosystem’, in *Journal of physics: Conference series*, IOP Publishing, vol. 608, 2015, p. 012 016.

[11] V.-A. Zamfir, M. Carabas, C. Carabas and N. Tapus, ‘Systems monitoring and big data analysis using the elasticsearch system’, in *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, IEEE, 2019, pp. 188–193.

[12] M. S. Divya and S. K. Goyal, ‘Elasticsearch: An advanced and quick search technique to handle voluminous data’, *Compusoft*, vol. 2, no. 6, p. 171, 2013.

[13] C. Gormley and Z. Tong, *Elasticsearch: The definitive guide: A distributed real-time search and analytics engine.* " O’Reilly Media, Inc.", 2015.

[14] G. Amato, P. Bolettieri, F. Carrara, F. Falchi and C. Gennaro, ‘Large-scale image retrieval with elasticsearch’, in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, ACM, 2018, pp. 925–928.

[15] O. Kononenko, O. Baysal, R. Holmes and M. W. Godfrey, ‘Mining modern repositories with elasticsearch’, in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 2014, pp. 328–331.

[16] R. Kuc and M. Rogozinski, *Elasticsearch server.* Packt Publishing Ltd, 2013.

[17] (2017). Basic concepts, [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/6.2/_basic_concepts.html (visited on 08/10/2019).

[18] J. Bai, ‘Feasibility analysis of big log data real time search based on hbase and elasticsearch’, in *2013 ninth international conference on natural computation (ICNC)*, IEEE, 2013, pp. 1166–1170.

[19] N. N. M. R. Subhani Shaik, ‘A review of elastic search: Performance metrics and challenges’, *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 5, pp. 222–229, 2017.

[20] U. Thacker, M. Pandey and S. S. Rautaray, ‘Performance of elasticsearch in cloud environment with ngram and non-ngram indexing’, in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, IEEE, 2016, pp. 3624–3628.

[21] (2019). Basic concepts, [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-forcemerge.html (visited on 08/10/2019).

[22] (2019). Tune for search speed, [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/master/tune-for-search-speed.html (visited on 08/10/2019).

[23] (2019). Tune for indexing speed, [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/master/tune-for-indexing-speed.html (visited on 11/10/2019).

[24] J. Turnbull, *The Logstash Book*. James Turnbull, 2013.

[25] H. Akshaya, J. Vidya and K. Veena, 'A basic introduction to devops tools', *International Journal of Computer Science & Information Technologies*, vol. 6, no. 3, pp. 05–06, 2015.

[26] W. Takase, T. Nakamura, Y. Watase and T. Sasaki, 'A solution for secure use of kibana and elasticsearch in multi-user environment', *arXiv preprint arXiv:1706.10040*, 2017.

[27] H. Waagsnes and N. Ulltveit-Moe, 'Intrusion detection system test framework for scada systems.', in *ICISSP*, 2018, pp. 275–285.

[28] J. Pokorny, 'Nosql databases: A step to database scalability in web environment', *International Journal of Web Information Systems*, vol. 9, no. 1, pp. 69–82, 2013.

[29] K. Banker, *MongoDB in action*. Manning Publications Co., 2011.

[30] G. Wang and J. Tang, 'The nosql principles and basic application of cassandra model', in *2012 International Conference on Computer Science and Service System*, IEEE, 2012, pp. 1332–1335.

[31] J. C. Anderson, J. Lehnardt and N. Slater, *CouchDB: the definitive guide: time to relax*. " O'Reilly Media, Inc.", 2010.

[32] S. Gupta and R. Rani, 'A comparative study of elasticsearch and couchdb document oriented databases', in *2016 International Conference on Inventive Computation Technologies (ICICT)*, IEEE, vol. 1, 2016, pp. 1–4.

[33] P. Seda, J. Hosek, P. Masek and J. Pokorny, 'Performance testing of nosql and rdbms for storing big data in e-applications', in *2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG)*, IEEE, 2018, pp. 1–4.

[34] U. Taware and N. Shaikh, 'Heterogeneous database system for faster data querying using elasticsearch', in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, IEEE, 2018, pp. 1–4.

[35] G. Tassey, 'The economic impacts of inadequate infrastructure for software testing', *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, pp. 429–489, 2002.

[36] J. D. Strate and P. A. Laplante, 'A literature review of research in software defect reporting', *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 444–454, 2013.

[37] G. R. Perez, G. Robles and J. M. G. Barahona, 'How much time did it take to notify a bug? two case studies: Elasticsearch and nova', in *2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM)*, IEEE, 2017, pp. 29–35.

[38] (2019). Esrally benchmarking, [Online]. Available: https://github.com/elastic/rally (visited on 09/09/2019).

[39] (2019). Esrally documentation, [Online]. Available: https://esrally.readthedocs.io/en/stable/ (visited on 08/09/2019).

[40] (2019). Docker documentation, [Online]. Available: https://docs.docker.com/ (visited on 05/10/2019).

[41] H. Robbins and S. Monro, 'A stochastic approximation method. in herbert robbins selected papers', 1985, pp. 102–109.

[42] Z. B. Zabinsky, *Stochastic adaptive search for global optimization.* Springer Science & Business Media, 2013, vol. 72.

[43] J. Brownlee, 'Clever algorithms', *Nature-Inspired Programming Recipes*, vol. 436, 2011.

[44] J. C. Spall *et al.*, 'Multivariate stochastic approximation using a simultaneous perturbation gradient approximation', *IEEE transactions on automatic control*, vol. 37, no. 3, pp. 332–341, 1992.

[45] J. Kiefer, J. Wolfowitz *et al.*, 'Stochastic estimation of the maximum of a regression function', *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[46] H. J. Kushner and D. S. Clark, *Stochastic approximation methods for constrained and unconstrained systems*. Springer Science & Business Media, 2012, vol. 26.

[47] H. Zhao and T. Liu, 'A parallelized combined direction simultaneous perturbation stochastic approximation algorithm', in *2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, IEEE, 2017, pp. 141–144.

[48] Z. Xu, 'A combined direction stochastic approximation algorithm', *Optimization Letters*, vol. 4, no. 1, pp. 117–129, 2010.

[49] R. Bløtekjær, 'Performance tuning apache drill on hadoop clusters with evolutionary algorithms', Master's thesis, 2018.

[50] G. Yildirim, İ. R. Hallac, G. Aydin and Y. Tatar, 'Running genetic algorithms on hadoop for solving high dimensional optimization problems', in *2015 9th International Conference on Application of Information and Communication Technologies (AICT)*, IEEE, 2015, pp. 12–16.

[51] S. F. Rodd and U. P. Kulkarni, 'Adaptive self-tuning techniques for performance tuning of database systems: A fuzzy-based approach', in *2013 2nd International Conference on Advanced Computing, Networking and Security*, IEEE, 2013, pp. 124–129.

[52] S. Kumar, S. Padakandla, P. Parihar, K. Gopinath, S. Bhatnagar *et al.*, 'Performance tuning of hadoop mapreduce: A noisy gradient approach', *arXiv preprint arXiv:1611.10052*, 2016.

[53] M. A. Rahman, J. Hossen and C. Venkataseshaiah, 'Smbsp: A self-tuning approach using machine learning to improve performance of spark in big data processing', in *2018 7th International Conference on Computer and Communication Engineering (ICCCE)*, IEEE, 2018, pp. 274–279.

Appendices

Appendix A

Docker-compose.yml

```yaml
version: '3.3'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.3.0
    container_name: elasticsearch1
    environment:
      - node.name=elasticsearch1
      - cluster.name=docker-cluster
      - cluster.initial_master_nodes=elasticsearch1
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms16G -Xmx16G"
      - http.cors.enabled=true
      - http.cors.allow-origin=*
      - network.host=_eth0_
    ulimits:
      nproc: 65535
      memlock:
        soft: -1
        hard: -1
    cap_add:
      - ALL
    # privileged: true
    deploy:
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      resources:
```

```
        limits:
          cpus: '1'
          memory: 256M
        reservations:
          cpus: '1'
          memory: 256M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 10s
    volumes:
      - type: volume
        source: logs
        target: /var/log
      - type: volume
        source: esdata1
        target: /usr/share/elasticsearch/
    networks:
      - elastic
      - ingress
    ports:
      - 9200:9200
      - 9300:9300
  elasticsearch2:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.3.0
    container_name: elasticsearch2
    environment:
      - node.name=elasticsearch2
      - cluster.name=docker-cluster
      - cluster.initial_master_nodes=elasticsearch1
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms16GM -Xmx16GM"
      - "discovery.zen.ping.unicast.hosts=elasticsearch1"
      - http.cors.enabled=true
      - http.cors.allow-origin=*
      - network.host=_eth0_
    ulimits:
      nproc: 65535
      memlock:
        soft: -1
        hard: -1
    cap_add:
      - ALL
    # privileged: true
    deploy:
```

```
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      resources:
        limits:
          cpus: '1'
          memory: 256M
        reservations:
          cpus: '1'
          memory: 256M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 10s
    volumes:
      - type: volume
        source: logs
        target: /var/log
      - type: volume
        source: esdata2
        target: /usr/share/elasticsearch/
    networks:
      - elastic
      - ingress
    ports:
      - 9201:9200

  logstash:
    container_name: logstash
    build:
      context: .
      dockerfile: Dockerfile-logstash
    restart: always
    ports:
      - "5000:5000"
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
    networks:
      - elastic
      - ingress

  kibana:
    image: docker.elastic.co/kibana/kibana:7.3.0
    container_name: kibana
```

```yaml
    environment:
      ELASTICSEARCH_HOSTS: http://128.39.120.25:9200/
    ports:
      - 5601:5601
    volumes:
      - type: volume
        source: logs
        target: /var/log
    ulimits:
      nproc: 65535
      memlock:
        soft: -1
        hard: -1
    cap_add:
      - ALL
    deploy:
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      resources:
        limits:
          cpus: '1'
          memory: 256M
        reservations:
          cpus: '1'
          memory: 256M
      restart_policy:
        condition: on-failure
        delay: 30s
        max_attempts: 3
        window: 120s
    networks:
      - elastic
      - ingress

  filebeat:
    image: docker.elastic.co/beats/filebeat:7.3.0
    command: --strict.perms=false
    environment:
      - setup.kibana.host=kibana:5601
      - output.elasticsearch.hosts=["elasticsearch:9200"]
    ports:
      - 9000:9000
    volumes:
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
```

```yaml
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - elastic


volumes:
  esdata1:
  esdata2:
  logs:

networks:
  elastic:
  ingress:
```

Appendix B

Data Generator

```
#!/usr/local/bin/python3.6
import json
from datetime import datetime
from elasticsearch import Elasticsearch
from elasticsearch import helpers
from json import dumps
import os, uuid
import time
from threading import  Thread
from mimesis import *
import sys
import logging


#Defining variables
person = Person()
food = Food()
unit_system = UnitSystem()
hardware = Hardware()
internet = Internet()
file = File()
address = Address()


# this variable will be used with the index name
date_today = datetime.today().strftime('%Y-%m-%d-%H.%M.%S')

# take input of elasticsearch IP and Port
```

```
host = "localhost"
port = 9200
number_of_documents = 0
# connect to the elasticsearch
client=Elasticsearch([{'host':host,'port':port}])



def enable_stdout(value):
    if (value):
        # log everything to stdout
        root = logging.getLogger()
        root.setLevel(logging.DEBUG)
        handler = logging.StreamHandler(sys.stdout)
        handler.setLevel(logging.DEBUG)
        formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
        handler.setFormatter(formatter)
        root.addHandler(handler)




class Fake_data:

    # open file for storing some info
    logfile = open("log.txt", "a")

    #create number of ojbects for Person
    def person_data(self,length):
        for i in range(length):
            yield     {
            #"_id": str(uuid.uuid4()),
            "_index": "person."+date_today,
            "doc": {
                "last_name": person.surname(),
                "first_name": person.name(),
                "Academic_degree": person.academic_degree(),
                "email": person.email()
            }
        }

    #create number of ojbects for Address
    def address_data(self,length):
        for i in range(length):
            yield     {
```

```
            #"_id": str(uuid.uuid4()),
            "_index": "address."+date_today,
            "doc": {
                "address": address.address(),
                "country_code": address.country_code(),
                "country": address.country(),
                "continent":address.continent(),
                "city":address.city(),
                "current_locale":address.get_current_locale(),
                "coordinates":address.coordinates(),
                "latitude":address.latitude(),
                "longitude": address.longitude()
            }
        }


    def file_data(self,length):
        for i in range(length):
            yield     {
            #"_id": str(uuid.uuid4()),
            "_index": "file."+date_today,
            "doc": {
                "file_name": file.file_name(),
                "size": file.size(),
                "mime_type": file.mime_type(),
                "extension":file.extension(),
            }
        }

    def hardware_data(self,length):
        for i in range(length):
            yield     {
            #"_id": str(uuid.uuid4()),
            "_index": "hardware."+date_today,
            "doc": {
                "cpu": hardware.cpu(),
                "frequency": hardware.cpu_frequency(),
                "codename": hardware.cpu_codename(),
                "model_code":hardware.cpu_model_code(),
                "generation":hardware.generation(),
                "manufacturer":hardware.manufacturer(),
                "graphics":hardware.graphics(),
                "phone_model":hardware.phone_model(),
                "ram_size": hardware.ram_size(),
                "ram_type": hardware.ram_type(),
                "resolution": hardware.resolution(),
```

```
                "ssd_or_hdd": hardware.ssd_or_hdd(),
                "screen_size": hardware.screen_size()
            }
        }

    def internet_data(self,length):
        for i in range(length):
            yield    {
            #"_id": str(uuid.uuid4()),
            "_index": "internet."+date_today,
            "doc": {
                "content_type": internet.content_type(),
                "ip": internet.ip_v4(),
                "ip_v6": internet.ip_v6(),
                "emoji":internet.emoji(),
                "home_page":internet.home_page(),
                "network_protocl":internet.network_protocol(),
                "mac_address":internet.mac_address(),
                "user_agent":internet.user_agent(),
                "port": internet.port(),
                "http_method": internet.http_method(),
                "http_status_code": internet.http_status_code(),
                "http_status_message": internet.http_status_message
    ()

            }
        }


    def unit_system_data(self,length):
        for i in range(length):
            yield    {
            #"_id": str(uuid.uuid4()),
            "_index": "unit_system."+date_today,
            "doc": {
                "unit": unit_system.unit(),
                "prefix": unit_system.prefix(),
            }
        }

    def food_data(self,length):
        for i in range(length):
            yield    {
            #"_id": str(uuid.uuid4()),
            "_index": "food."+date_today,
            "doc": {
```

```
                "vegetable": food.vegetable(),
                "fruit": food.fruit(),
                "dish": food.dish(),
                "drink":food.drink(),
                "spices":food.spices(),
                "current_locale":food.get_current_locale(),
            }
        }

    def generate_all_types(self,length=500000):
        start = time.time()
        try:
            Thread(target=helpers.bulk, args=[client, [i for i in self.person_data(length)]]).start()
            Thread(target=helpers.bulk, args=[client, [i for i in self.internet_data(length)]]).start()
            Thread(target=helpers.bulk, args=[client, [i for i in self.file_data(length)]]).start()
            Thread(target=helpers.bulk, args=[client, [i for i in self.food_data(length)]]).start()
            Thread(target=helpers.bulk, args=[client, [i for i in self.hardware_data(length)]]).start()
            Thread(target=helpers.bulk, args=[client, [i for i in self.unit_system_data(length)]]).start()
            Thread(target=helpers.bulk, args=[client, [i for i in self.address_data(length)]]).start()


            end =time.time() - start

            self.logfile.write(date_today+" : The proess to bulk "+ str(length*7) + " documents took "+ str(end) + " seconds"+'\n')
            self.logfile.close()
        except Exception as e:
            self.logfile.write(date_today+ " : "+ str(e)+"\n")
            self.logfile.close()



class __main__:
    def main():
        global host
        global port
        global client
        global number_of_documents
```

```

    if len(sys.argv) >= 3:
        host = sys.argv[1]
        port = sys.argv[2]
        number_of_documents = sys.argv[3]
        with_stdout = sys.argv[4].lower() == 'true'

    enable_stdout(with_stdout)
    client=Elasticsearch([{'host':host,'port':port}])
    generator = Fake_data()
    generator.generate_all_types(int(number_of_documents))

if __name__ == "__main__":
    main()
```

Appendix C

ESRally Connector

```
import json
from datetime import datetime
from elasticsearch import Elasticsearch
from elasticsearch import helpers
from json import dumps
import os, uuid
import time
from threading import  Thread
import requests



class ESRally_connector(object):
    datetime=datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
    dir_name = "rally"+ str(datetime)
    command = ""
    def __init__(self, elasticsearch_node):
        self.elasticsearch_node = elasticsearch_node
        self.command = "sudo docker run --rm -v $PWD/"+self.dir_name+":/rally/.rally elastic/rally --track=nyc_taxis --test-mode --pipeline=benchmark-only --target-hosts="+self.elasticsearch_node
        #128.39.120.25:9200

    def get_race_json(self):
        print("Creating new directory to store the test ...")
        os.system('mkdir '+ self.dir_name)
        os.system('sudo chgrp 0 $PWD/'+self.dir_name)
```

```
        os.system(self.command)
        os.system(self.command)
        f = [(os.getcwd()+"/"+self.dir_name+"/benchmarks/races/"+dI) for dI in os.listdir(self.dir_name+'/benchmarks/races/') if os.path.isdir(os.path.join(self.dir_name+'/benchmarks/races/',dI))]
        os.chdir(str(f[0]))
        print('path of race.json : '+str(f[0])+"/race.json")
        return str(f[0])+"/race.json"

    def update_node_settings(self,RF,TS,SI,RMB):
    ## in future, make it read a list
        referesh_interval = "5"
        url = "http://"+self.elasticsearch_node+"/_settings"
        data = {'index' : {'refresh_interval' : RF,'translog' : {"flush_threshold_size": TS, "sync_interval": SI }}}
        headers = {'Content-type': 'application/json'}
        r = requests.put(url, data=json.dumps(data), headers=headers)

        ##updating cluster parameters
        url_cluster = "http://"+self.elasticsearch_node+"/_cluster/settings"
        data_rmb = {"persistent" : {"indices.recovery.max_bytes_per_sec" : RMB}}
        cluster_request = requests.put(url, data=json.dumps(data), headers=headers)

        print(r)
        print(cluster_request)
```

Appendix D

Parameters

```
import math

class Elasticsearch_Parameters:
    def __init__(self,_index_refresh_interval=10,
    _index_translog_flush_threshold_size=300,
    _translog_sync_interval=100,_recovery_max_bytes_per_sec=50):
            self._index_refresh_interval = _index_refresh_interval
            self._index_translog_flush_threshold_size =
    _index_translog_flush_threshold_size
            self._translog_sync_interval = _translog_sync_interval
            self._recovery_max_bytes_per_sec =
    _recovery_max_bytes_per_sec
     # using property decorator
     # a getter function

#################### index.referesh_interval
    @property
    def index_refresh_interval(self):
         return self._index_refresh_interval

    @index_refresh_interval.setter
    def index_refresh_interval(self, value):
        if (value > self.minmax_index_refresh_interval()[1]):
            self._index_refresh_interval = self.
    minmax_index_refresh_interval()[1]
        if (value < self.minmax_index_refresh_interval()[0]):
            self._index_refresh_interval = self.
    minmax_index_refresh_interval()[0]
```

```
        else:
            self._index_refresh_interval = value

    def index_refresh_interval_string(self):
         return (str(self._index_refresh_interval) + "s")

    def minmax_index_refresh_interval(self):
        return [1,8000]

    def minmax_index_refresh_interval_current(self,current):
        self._index_refresh_interval = current
        return [1,8000,self._index_refresh_interval]

    def return_minmax_index_refresh_interval_current(self,):
        return [1,8000,self._index_refresh_interval]

    def set_scale_index_refresh_interval(self,scale):
        self.scale_index_refresh_interval = scale
    def scale_index_refresh_interval(self):
        return self.scale_index_refresh_interval

    def scale_minmax_index_refresh_interval(self):
        minmax = self.minmax_index_refresh_interval()
        min = math.ceil((minmax[1]-minmax[0] )/100)
        max = math.ceil((minmax[1]-minmax[0] )/10)
        return [min,max]

    def calculate_step_size_refresh_interval(self, improvement_percentage):
        res = self.scale_minmax_index_refresh_interval()[0] * (1 + improvement_percentage)
        if res < self.scale_minmax_index_refresh_interval()[1]:
            return res
        else:
            return self.scale_minmax_index_refresh_interval()[1]


##################### index.flush_threshold_size
    @property
    def index_translog_flush_threshold_size(self):
         return self._index_translog_flush_threshold_size

    @index_translog_flush_threshold_size.setter
    def index_translog_flush_threshold_size(self, value):
        if (value > self.minmax_index_translog_flush_threshold_size()[1]):
```

```
            self._index_translog_flush_threshold_size = self.minmax_index_translog_flush_threshold_size()[1]
        if (value < self.minmax_index_translog_flush_threshold_size()[0]):
            self._index_translog_flush_threshold_size = self.minmax_index_translog_flush_threshold_size()[0]
        else:
            self._index_translog_flush_threshold_size = value

    def index_translog_flush_threshold_size_string(self):
         return (str(self._index_translog_flush_threshold_size) + "mb")

    def minmax_index_translog_flush_threshold_size(self):
         return [112,10000]

    def minmax_index_translog_flush_threshold_size_current(self, current):
        self._index_translog_flush_threshold_size = current
        return [112,10000,self._index_translog_flush_threshold_size]

    def return_minmax_index_translog_flush_threshold_size_current(self,):
        return [112,10000,self._index_translog_flush_threshold_size]

    def set_scale_index_translog_flush_threshold_size(self,scale):
        self.scale_index_translog_flush_threshold_size = scale
    def scale_index_translog_flush_threshold_size(self):
        return self.scale_index_translog_flush_threshold_size

    def scale_minmax_index_translog_flush_threshold_size(self):
        minmax = self.minmax_index_translog_flush_threshold_size()
        min = math.ceil((minmax[1]-minmax[0] )/100)
        max = math.ceil((minmax[1]-minmax[0] )/10)
        return [min,max]

    def calculate_step_size_threshold_size(self, improvement_percentage):
        res = self.scale_minmax_index_translog_flush_threshold_size()[0] * (1 + improvement_percentage)
        if res < self.scale_minmax_index_translog_flush_threshold_size()[1]:
            return res
        else:
```

```
            return self.
    scale_minmax_index_translog_flush_threshold_size()[1]


################################## translog_sync_interval
    @property
    def translog_sync_interval(self):
         return self._translog_sync_interval

    @translog_sync_interval.setter
    def translog_sync_interval(self, value):
        if (value > self.minmax_translog_sync_interval()[1]):
            self._translog_sync_interval = self.
    minmax_translog_sync_interval()[1]
        if (value < self.minmax_translog_sync_interval()[0]):
            self._translog_sync_interval = self.
    minmax_translog_sync_interval()[0]
        else:
            self._translog_sync_interval = value

    def translog_sync_interval_string(self):
         return (str(self._translog_sync_interval) + "s")

    def minmax_translog_sync_interval(self):
         return [1,10000]

    def minmax_translog_sync_interval_current(self,current):
        self._translog_sync_interval = current
        return [1,10000,self._translog_sync_interval]

    def return_minmax_translog_sync_interval_current(self,):
        return [1,10000,self._translog_sync_interval]

    def set_scale_translog_sync_interval(self,scale):
        self.scale_translog_sync_interval = scale
    def scale_translog_sync_interval(self):
        return self.scale_translog_sync_interval

    def scale_minmax_translog_sync_interval(self):
        minmax = self.minmax_translog_sync_interval()
        min = math.ceil((minmax[1]-minmax[0] )/100)
        max = math.ceil((minmax[1]-minmax[0] )/10)
        return [min,max]

    def calculate_step_size_translog_sync_interval(self,
    improvement_percentage):
```

```
        res = self.scale_minmax_translog_sync_interval()[0] * (1 + improvement_percentage)
        if res < self.scale_minmax_translog_sync_interval()[1]:
            return res
        else:
            return self.scale_minmax_index_translog_sync_interval()[1]

################################indices.recovery.max_bytes_per_sec

    @property
    def recovery_max_bytes_per_sec(self):
         return self._recovery_max_bytes_per_sec

    @recovery_max_bytes_per_sec.setter
    def recovery_max_bytes_per_sec(self, value):
        if (value > self.minmax_recovery_max_bytes_per_sec()[1]):
            self._recovery_max_bytes_per_sec = self.minmax_recovery_max_bytes_per_sec()[1]
        if (value < self.minmax_recovery_max_bytes_per_sec()[0]):
            self._recovery_max_bytes_per_sec = self.minmax_recovery_max_bytes_per_sec()[0]
        else:
            self._recovery_max_bytes_per_sec = value

    def recovery_max_bytes_per_sec_string(self):
         return (str(self._recovery_max_bytes_per_sec) + "mb")

    def minmax_recovery_max_bytes_per_sec(self):
         return [50,10000]

    def minmax_recovery_max_bytes_per_sec_current(self,current):
        self._recovery_max_bytes_per_sec = current
        return [50,10000,self._recovery_max_bytes_per_sec]

    def return_minmax_recovery_max_bytes_per_sec_current(self):
        return [50,10000,self._recovery_max_bytes_per_sec]

    def set_scale_recovery_max_bytes_per_sec(self,scale):
        self.scale_recovery_max_bytes_per_sec = scale

    def scale_recovery_max_bytes_per_sec(self):
        return self.scale_recovery_max_bytes_per_sec

    def scale_minmax_recovery_max_bytes_per_sec(self):
        minmax = self.minmax_recovery_max_bytes_per_sec()
```

```
        min = math.ceil((minmax[1]-minmax[0] )/100)
        max = math.ceil((minmax[1]-minmax[0] )/10)
        return [min,max]

    def calculate_step_size_recovery_max_bytes_per_sec(self,
    improvement_percentage):
        res = self.scale_minmax_recovery_max_bytes_per_sec()[0] *
    (1 + improvement_percentage)
        if res < self.scale_minmax_recovery_max_bytes_per_sec()[1]:
            return res
        else:
            return self.scale_minmax_recovery_max_bytes_per_sec()
    [1]
```

Appendix E

Perturbation Optimizer

```
import random
from race_reader import race_reader
from Parameters import Elasticsearch_Parameters
import ESRally_Connector
import logging
import math
from Plotting import Plotting

class Optimizer(object):
    """docstring for Optimizer."""

    def __init__(self, Parameters_object):
        super(Optimizer, self).__init__()

        # define the elasticsearch ip and port to connect esrally
        self.connector = ESRally_Connector.ESRally_connector("
    128.39.120.25:9200")
        self.par = Parameters_object

        # define a plotting object
        self.plotting = Plotting()

        # creates a logger
        self.logger = logging.getLogger(__name__)
        self.logger.setLevel(logging.INFO)
        self.file_handler = logging.FileHandler('tuning_process.log
    ')
        self.formatter = logging.Formatter('%(asctime)s : %(
```

```
levelname)s : %(message)s')
        self.file_handler.setFormatter(self.formatter)
        self.logger.addHandler(self.file_handler)

    def objective_function(self, indexing, response_time):
        self.plotting.add_latency_value(response_time)
        self.plotting.add_indexing_value(indexing)
        self.plotting.add_index_latency((response_time,indexing))
        return indexing/response_time, [response_time,indexing]


    def next_vector_value(self, minmax):
        i = 0
        limit = len(minmax)
        vector = [0 for i in range(limit)]

        # get scaling values in a list
        scale = [self.par.scale_index_refresh_interval,
                 self.par.scale_index_translog_flush_threshold_size]
        random_operator = [random.choice((-1, 1)) for _ in range(limit)]
        chance = [a*b for a, b in zip(scale, random_operator)]
        print("the X+ = ", chance)
        X_negative = [i * -1 for i in chance]  # get X- by multply -1 to X+
        print("the X- = ", X_negative)

        #####
        for i in range(limit):
            vector[i] = self.next(minmax[i], chance[i])

        return vector

    def next(self, current_par, chance):
        # increase or decrease value of a pramaeter
        if (((current_par[2] + chance) <= current_par[1]) or ((current_par[2] + chance) >= current_par[0])):
            return (current_par[2] + chance)


    def optimizer(self,current_par, max_iter):
        best_of_best = {}

        best = {}
        candidate = {}
```

```
        self.x_plus_minus_improvement = 0

        self.current_par = current_par
        self.logger.info('Set of Parameters = ' + str(current_par))

        for i in range(max_iter):
            X_positive, X_negative = self.x_plus_minus(self.current_par)
            candidate['vector_positive'] = X_positive
            candidate['vector_negative'] = X_negative

            self.logger.info('X+ Parameters '+ str(X_positive))
            self.logger.info('X- Parameters '+ str(X_negative))

            print("Vector_Positive : ", str(candidate['vector_positive']))
            print("vector_negative : ", str(candidate['vector_negative']))

            # get objective function of the X+ and X-
            of_plus = self.takes_vector(candidate['vector_positive'])[0]
            self.plotting.add_dot_data((i+1,float(of_plus)))

            print("OF of X+ = ", of_plus)
            self.logger.info('Objective function of X+ = ' + str(of_plus))

            of_minus = self.takes_vector(candidate['vector_negative'])[0]
            self.plotting.add_dot_data((i+1,float(of_minus)))


            print("OF of X- = ", of_minus)
            self.logger.info('Objective function of X- = ' + str(of_minus))

            # get the improvement percentage of of+ and of-
            self.x_plus_minus_improvement = self.calculate_percentage_difference(float(of_plus), float(of_minus))

            print("calculate_percentage_difference : ", self.x_plus_minus_improvement)
            self.logger.info('calculate percentage difference = ' +
```

```
 str(self.x_plus_minus_improvement))
        # find which of (x+ x-) is best
        self.logger.info('choosing which direction to go  x+ or x- ')

        if (of_plus > of_minus):
            best['cost'] = of_plus
            best['vector'] = candidate['vector_positive']
            self.logger.info('X+ is better than X- with objective_function = '+ str(best['cost']))

            # add line plotting to the graph
            self.plotting.add_line_data((i+1,float(of_plus)))

            self.update_parameters(best)
            self.logger.info('setting new parameters value based on X+ which is '+ str(best['vector']))
        else:
            best['cost'] = of_minus
            best['vector'] = candidate['vector_negative']
            self.logger.info('X- is better than X+ with objective_function = '+ str(best['cost']))

            # add line plotting to the graph
            self.plotting.add_line_data((i+1,float(of_minus)))


            self.update_parameters(best)
            self.logger.info('setting new parameters value based on X- which is '+ str(best['vector']))

        if not best_of_best or best['cost'] > best_of_best['cost']:
            best_of_best['cost'] = best.get('cost')
            best_of_best['vector'] = best.get('vector')
            self.logger.info('updating best_of_best dic with value :::: '+ str(best_of_best))

        #setting the new list of [min,max,current]
        self.current_par = [self.par.minmax_index_refresh_interval_current(self.par.index_refresh_interval),
        self.par.minmax_index_translog_flush_threshold_size_current(self.par.index_translog_flush_threshold_size),self.par.minmax_translog_sync_interval_current(self.par.
```

```
translog_sync_interval),
            self.par.minmax_recovery_max_bytes_per_sec_current(self.par.recovery_max_bytes_per_sec)]

            # update the scaling values by multiplying to the percentage difference
            self.logger.info('Updating the step size')
            self.update_step_size()
            #self.update_scale()

            scale = [self.par.scale_index_refresh_interval,self.par.scale_index_translog_flush_threshold_size,self.par.scale_translog_sync_interval,
            self.par.recovery_max_bytes_per_sec]
            print("after updating scaling values : ", scale)
            self.logger.info('The new scaling values for each parameter in order '+ str(scale))

        self.logger.info('returning Best  '+ str(best_of_best))
        self.logger.info("Line_data '{0}' and dots_data '{1}'".format(str(self.plotting.get_line_dots()[0]),str(self.plotting.get_line_dots()[1])))
        self.logger.info("indexing data and latency" + str(self.plotting.get_indexing_value())+" ########## "+str(self.plotting.get_latency_value()))
        self.logger.info("indexing data and latency '{0}'".format(str(self.plotting.get_latency_value())))
        return best_of_best

    def x_plus_minus(self, minmax):
        i = 0
        limit = len(minmax)
        vector_x_plus = [0 for i in range(limit)]
        vector_x_minus = [0 for i in range(limit)]
        # get scaling values in a list
        scale = [self.par.scale_index_refresh_interval,self.par.scale_index_translog_flush_threshold_size,self.par.scale_translog_sync_interval,self.par.scale_recovery_max_bytes_per_sec]
        # make a random + or - to each parameter
        random_operator = [random.choice((-1, 1)) for _ in range(limit)]
        X_positive = [a*b for a, b in zip(scale, random_operator)]
        # get X- by multply -1 to X+
        X_negative = [i * -1 for i in X_positive]

```

```
        for i in range(limit):
            vector_x_plus[i] = self.next(minmax[i], X_positive[i])
            vector_x_minus[i] = self.next(minmax[i], X_negative[i])

        # return x+ and x- in a list
        return [vector_x_plus, vector_x_minus]

    def calculate_percentage_difference(self, v1, v2):
        return ((abs(v1 - v2) / ((v1+v2)/2)) * (100.0))/100

    def update_parameters(self,list):
        self.par.index_refresh_interval = list['vector'][0]
        self.par.index_translog_flush_threshold_size = list['vector'][1]
        self.par.translog_sync_interval = list['vector'][2]
        self.par.recovery_max_bytes_per_sec = list['vector'][3]

    def update_scale(self):
        ##### refresh interval
        # get the expected next par value and check if it's within minmax range
        minmax_current_rf = self.par.return_minmax_index_refresh_interval_current()
        rf_tmp_plus = math.ceil(((self.par.scale_index_refresh_interval * self.x_plus_minus_improvement)+self.par.scale_index_refresh_interval) + minmax_current_rf[2])
        rf_tmp_minus= math.ceil((-1*(self.par.scale_index_refresh_interval * self.x_plus_minus_improvement))+self.par.scale_index_refresh_interval + minmax_current_rf[2])
        rf_mx = minmax_current_rf[1]
        rf_min = minmax_current_rf[0]

        self.logger.info('update_scale() minmax_current_rf, rf_tmp_plus ,rf_tmp_minus '+ str(minmax_current_rf)+ str(rf_tmp_plus)+ str(rf_tmp_minus))
        if rf_min <= rf_tmp_plus <= rf_mx and rf_min <= rf_tmp_minus <= rf_mx :
            self.par.set_scale_index_refresh_interval(math.ceil((self.par.scale_index_refresh_interval * self.x_plus_minus_improvement)+self.par.scale_index_refresh_interval))
        else:
            self.logger.info('new scale value exceeds the minmax range. The same scaling value will be used for RI ' + str(self.par.scale_index_refresh_interval))

```

```python
        ##### threshold size
        minmax_current_ts = self.par.return_minmax_index_translog_flush_threshold_size_current()
        ts_tmp_plus = math.ceil((self.par.scale_index_translog_flush_threshold_size * self.x_plus_minus_improvement+self.par.scale_index_translog_flush_threshold_size)+ minmax_current_ts[2])
        ts_tmp_minus= math.ceil(((self.par.scale_index_translog_flush_threshold_size * self.x_plus_minus_improvement)*(-1))+self.par.scale_index_translog_flush_threshold_size + minmax_current_ts[2])
        ts_mx = minmax_current_ts[1]
        ts_min = minmax_current_ts[0]

        if ts_min <= ts_tmp_plus <= ts_mx and ts_min <= ts_tmp_minus <= ts_mx:
            self.par.set_scale_index_translog_flush_threshold_size(math.ceil((self.par.scale_index_translog_flush_threshold_size * self.x_plus_minus_improvement))+self.par.scale_index_translog_flush_threshold_size)
        else:
            self.logger.info('new scale value exceeds the minmax range. The same scaling value will be used for ' + str(self.par.scale_index_translog_flush_threshold_size))


        ##### recovery_max_bytes_per_sec
        minmax_current_rmb = self.par.return_minmax_recovery_max_bytes_per_sec_current()
        rmb_tmp_plus = math.ceil((self.par.scale_recovery_max_bytes_per_sec * self.x_plus_minus_improvement+self.par.scale_recovery_max_bytes_per_sec)+ minmax_current_rmb[2])
        rmb_tmp_minus= math.ceil(((self.par.scale_recovery_max_bytes_per_sec * self.x_plus_minus_improvement)*(-1))+self.par.scale_recovery_max_bytes_per_sec + minmax_current_rmb[2])
        rmb_mx = minmax_current_rmb[1]
        rmb_min = minmax_current_rmb[0]

        if rmb_min <= rmb_tmp_plus <= rmb_mx and rmb_min <= rmb_tmp_minus <= rmb_mx:
            self.par.set_scale_recovery_max_bytes_per_sec(math.ceil((self.par.scale_recovery_max_bytes_per_sec *self.
```

```
x_plus_minus_improvement))+self.par.scale_recovery_max_bytes_per_sec)
        else:
            self.logger.info('new scale value exceeds the minmax range. The same scaling value will be used for ' + str(self.par.scale_recovery_max_bytes_per_sec))




    def update_step_size(self):
        ##### refresh interval
        # get the expected next par value and check if it's within minmax range
        minmax_current_rf = self.par.return_minmax_index_refresh_interval_current()
        rf_tmp_plus = math.ceil(self.par.calculate_step_size_refresh_interval(self.x_plus_minus_improvement)) + minmax_current_rf[2]
        rf_tmp_minus= math.ceil((-1*(self.par.calculate_step_size_refresh_interval(self.x_plus_minus_improvement))) + minmax_current_rf[2])
        rf_mx = minmax_current_rf[1]
        rf_min = minmax_current_rf[0]

        self.logger.info(' update_step_size() minmax_current_rf, rf_tmp_plus ,rf_tmp_minus '+ str(minmax_current_rf)+ str(rf_tmp_plus)+","+ str(rf_tmp_minus))
        if rf_min <= rf_tmp_plus <= rf_mx and rf_min <= rf_tmp_minus <= rf_mx :
            self.par.set_scale_index_refresh_interval(math.ceil(self.par.calculate_step_size_refresh_interval(self.x_plus_minus_improvement)))
        else:
            self.par.set_scale_index_refresh_interval(math.ceil(self.par.scale_index_refresh_interval*0.01))
            self.logger.info('new scale value exceeds the minmax range. The same scaling value will be used for RI ' + str(self.par.scale_index_refresh_interval))

        ##### threshold size
        minmax_current_ts = self.par.return_minmax_index_translog_flush_threshold_size_current()
        ts_tmp_plus = math.ceil(self.par.calculate_step_size_threshold_size(self.x_plus_minus_improvement))+ minmax_current_ts[2]
```

```
        ts_tmp_minus= math.ceil(-1*(self.par.calculate_step_size_threshold_size(self.x_plus_minus_improvement)) + minmax_current_ts[2])
        ts_mx = minmax_current_ts[1]
        ts_min = minmax_current_ts[0]

        if ts_min <= ts_tmp_plus <= ts_mx and ts_min <= ts_tmp_minus <= ts_mx:
            self.par.set_scale_index_translog_flush_threshold_size(math.ceil(self.par.calculate_step_size_threshold_size(self.x_plus_minus_improvement)))
        else:
            self.par.set_scale_index_translog_flush_threshold_size(math.ceil(self.par.scale_index_translog_flush_threshold_size*0.01))
            self.logger.info('new scale value exceeds the minmax range. The same scaling value will be used for TS ' + str(self.par.scale_index_translog_flush_threshold_size))


       ##### translog_sync_interval
        minmax_current_si = self.par.return_minmax_translog_sync_interval_current()
        si_tmp_plus = math.ceil(self.par.calculate_step_size_translog_sync_interval(self.x_plus_minus_improvement))+ minmax_current_si[2]
        si_tmp_minus= math.ceil(-1*(self.par.calculate_step_size_translog_sync_interval(self.x_plus_minus_improvement)) + minmax_current_si[2])
        si_mx = minmax_current_si[1]
        si_min = minmax_current_si[0]

        if si_min <= si_tmp_plus <= si_mx and si_min <= si_tmp_minus <= si_mx:
            self.par.set_scale_translog_sync_interval(math.ceil(self.par.calculate_step_size_translog_sync_interval(self.x_plus_minus_improvement)))
        else:
            self.par.set_scale_translog_sync_interval(math.ceil(self.par.scale_translog_sync_interval*0.01))
            self.logger.info('new scale value exceeds the minmax range. The same scaling value will be used for SI' + str(self.par.scale_translog_sync_interval))

        ##### recovery_max_bytes_per_sec
        minmax_current_rmb = self.par.
```

```
return_minmax_recovery_max_bytes_per_sec_current()
        rmb_tmp_plus = math.ceil(self.par.calculate_step_size_recovery_max_bytes_per_sec(self.x_plus_minus_improvement))+ minmax_current_si[2]
        rmb_tmp_minus= math.ceil(-1*(self.par.calculate_step_size_recovery_max_bytes_per_sec(self.x_plus_minus_improvement)) + minmax_current_si[2])
        rmb_mx = minmax_current_rmb[1]
        rmb_min = minmax_current_rmb[0]

        if rmb_min <= rmb_tmp_plus <= rmb_mx and rmb_min <= rmb_tmp_minus <= rmb_mx:
            self.par.set_scale_recovery_max_bytes_per_sec(math.ceil(self.par.calculate_step_size_recovery_max_bytes_per_sec(self.x_plus_minus_improvement)))
        else:
            self.par.set_scale_recovery_max_bytes_per_sec(math.ceil(self.par.scale_recovery_max_bytes_per_sec*0.1))
            self.logger.info('new scale value exceeds the minmax range. The same scaling value will be used for RMB' + str(self.par.scale_recovery_max_bytes_per_sec))


    def takes_vector(self, candidate):
        # take a set of parameters and perofrm the elasticsearch update and then return the value of the objective function
        self.par.index_refresh_interval = candidate[0]
        self.par.index_translog_flush_threshold_size = candidate[1]
        self.par.translog_sync_interval = candidate[2]
        self.par.recovery_max_bytes_per_sec = candidate[3]
        self.connector.update_node_settings(self.par.index_refresh_interval_string(), self.par.index_translog_flush_threshold_size_string(),self.par.index_translog_flush_threshold_size_string(),self.par.recovery_max_bytes_per_sec_string())
        self.logger.info('Updating Elasticsearch settings')
        self.myRace = race_reader(self.connector.get_race_json())
        self.logger.info('Start ESRally benchamrking ')
        return self.objective_function(self.myRace.index_throughput_mean(), self.myRace.ops_latency_average_mean())


    def make_graph(self):
        return self.plotting.plot_chart()
```

Appendix F

Race Reader

```python
import json

class race_reader(object):
    """
     Reading json files to return certain values
    """
    def __init__(self, json_file):
        with open(json_file) as js:
            self.json_file = json.load(js)

    def index_throughput_mean(self):
        return float(json.dumps(self.json_file["results"]["op_metrics"][0]["throughput"]["mean"]))

    def ops_latency_average_mean(self):
        #since the number of operations is the same for each type we take the average
        ops_len = len(self.json_file["results"]["op_metrics"])
        res = 0
        for i in range(1, ops_len):
            res =+ float(self.json_file["results"]["op_metrics"][i]["latency"]["mean"])
        return res / ops_len
```

Appendix G

Main.py

```
import ESRally_Connector
import race_reader
from perturbation_optimizer import Optimizer
from Parameters import Elasticsearch_Parameters


def main():
    #get init parameters
    par=Elasticsearch_Parameters(100,1500,100,7000)
    ri = par.index_refresh_interval
    fs = par.index_translog_flush_threshold_size
    si = par.translog_sync_interval
    mbr = par.recovery_max_bytes_per_sec
    # seting the scaling value of each parameter
    par.set_scale_index_refresh_interval(par.scale_minmax_index_refresh_interval()[0])
    par.set_scale_index_translog_flush_threshold_size(par.scale_minmax_index_translog_flush_threshold_size()[0])
    par.set_scale_translog_sync_interval(par.scale_minmax_translog_sync_interval()[0])
    par.set_scale_recovery_max_bytes_per_sec(par.scale_minmax_recovery_max_bytes_per_sec()[0])

    current_par = [par.minmax_index_refresh_interval_current(ri),par.minmax_index_translog_flush_threshold_size_current(fs),par.minmax_translog_sync_interval_current(si),par.minmax_recovery_max_bytes_per_sec_current(mbr)]

```

```
    search_space = current_par
    max_iter = 15

  # execute the algorithm
    best = Optimizer(par)
    #best = best.search(search_space ,max_iter)
    best = best.x(search_space ,max_iter)
    #Updating the settings from the best results
    self.connector.update_node_settings(best['cost'][0],best['cost'][1],best['cost'][2],best['cost'][3])
    best.make_graph()
    print("Done. Best Solution: cost = " + str(best['cost']) + ", v = " + str(best['vector']))


if __name__ == '__main__':
    main()
```

Appendix H

Plotting

```
import leather


class Plotting(object):

    def __init__(self):
        self.line_data = []
        self.dot_data = []
        self.latency = []
        self.indexing = []
        self.latency_index = []
        self.chart = leather.Chart('Results')

    def add_line_data(self, list):
        self.line_data.append(list)

    def add_dot_data(self, list):
        self.dot_data.append(list)

    def plot_chart(self):
        self.chart.add_line(self.line_data)
        self.chart.add_dots(self.dot_data)
        self.chart.to_svg('result.svg')
        #/home/elasticsearch

    def get_line_dots(self):
        return self.line_data, self.dot_data

```

```
    def add_latency_value(self,value):
        self.latency.append(value)

    def add_indexing_value(self,value):
        self.indexing.append(value)

    def get_latency_value(self):
        return self.latency

    def get_indexing_value(self):
        return self.indexing

    def add_index_latency(self,list):
        self.latency_index.append(list)

    def get_index_latency_value(self):
        return self.latency_index
```