TIAGO RODRIGO KEPE

SELF-TUNING BASED ON DATA SAMPLING

Master's dissertation presented to the Informatics Graduation Program, Federal University of Paraná.
Advisor: Prof. Dr. Eduardo C. de Almeida

CURITIBA

2013

CONTENTS

RESUMO

Texto do resumo....

ABSTRACT

Currently with the popularity of internet and phenomenon of the social networks a large amount of data is generated day-to-day. MapReduce appears as a powerful paradigm to analyse and process such amount of data. The Hadoop framework implements the MapReduce paradigm, in which a simple interface is available to implement MapReduce(MR) jobs. However, in MR jobs developers are allowed to setup several parameters to draw optimal performance from the available resources, but find a good configuration is time consuming and a configuration found in an execution may be impracticable for the next time.

In order to facilitate and automate tuning hadoop jobs, we propose a self-tuning based on data sampling. Our approach allows to find a good job configuration considering the data stored and the job in question. The users till can provide their usual job configurations then get the new job configuration that will be more appropriate with the current state of data stored and the hadoop cluster. So the users have end-to-end tool to automate the choice of knobs for each job.

CHAPTER 1

INTRODUCTION

In this chapter we present our motivation and objectives for this work, and we present the organization of the document.

1.1 Motivation

Nowadays the big companies are processing and generating a vast amount of data day-to-day. This companies are growing investing in distributed and parallel computing to process such data. To perform the distributed computing efficiently the data storage must be simple in order to allow parallel processing. The key-value model is a possible solution to build applications to data distributed processing, e.g. the MR programing paradigm which is based on key-value model[9].

MapReduce became the industry de facto standard for parallel processing. Attractive features such as scalability and reliability motivate many large companies such as Facebook, Google, Yahoo and research institutes to adopt this new programming paradigm. Key-value model and MR paradigm are implemented on the framework Hadoop, an open-source implementation of MapReduce, and these organizations rely on Hadoop [29] to process their information. Besides Hadoop, several other implementations are available: Greenplum MapReduce [14], Aster Data [2], Nokia Disco [22] and Microsoft Dryad [18].

MapReduce has a simplified programming model, where data processing algorithms are implemented as instances of two higher-order functions: Map and Reduce. All complex issues related to distributed processing, such as scalability, data distribution and reconciliation, concurrence and fault tolerance are managed by the framework. The main complexity that is left to the developer of a MapReduce-based application (also called a job) lies in the design decisions made to split the application specific algorithm into the two higher-order functions. Even if some decisions may result in a functionally correct

application, bad design choices might also lead to poor resource usage.

Implement jobs on Hadoop is simple, but there are many of knobs to adjust depending on the available resources (e.g. input data, online machines, network bandwidth, etc.) that improve the job performance. One relevant aspect is that the MR jobs are expected to work with large amounts of data, which can be the main barrier to find a good configuration [5]. Therefore, data sampling can be useful to improve on processing adjust of parameters instead of processing all data set how is done in [16]. But generate a representative and relevant data sampling is hard and a bad sampling may not represent several aspects related to the computation in large-scale: efficient resource usage, correct merge of data and intermediate data.

1.2 Objectives

Our objective is to propose a self-tuning based on data sampling over Hadoop, so we intend to use an evolutionary algorithm [3] to select good configurations for MR jobs. Based in our knowledge the best way to find such configurations is to run the jobs with its and analyse the performance, normally this process is done manually. But a crucial trouble is the large amount of data stored that can increase exponentially the test time of the job. One way to solve this trouble is to create a data sample. We propose one method to implement data sampling using key-value model and MR paradigm.

1.3 Contribution

We present an original approach to automate Hadoop job configuration, our approach is based on an bacteriological algorithm [3] in order to avoid run it on all data storage we develop one method to obtain data samples from hadoop input data. For data sampling we considered a lot of aspects related the paradigm MR, key-value model and others hadoop particularities. Our approach presents an user interface which through a domain specific language (**DSL**), to facilitate the user iteraction with data sampling and drive tuning procedures.

Our proposal intents to establish a framework to automate Hadoop job configuration, through the following proposals:

- an algorithm to automate a self-tuning;
- a method for sampling data on Hadoop clusters.
- an interface for users based on domain specific language;

As measure of performance, we used the latency time that the job led to conclude. Furthermore, we intend to use other measures of performance such as amount of intermidiate data, network usage and cpu usage.

1.4 Outline

- Chapter 2 introduces the fundamental concepts of the key-value model, MR paradigm and hadoop framework.
- Chapter 3 presents the bacteriological algorithm.
- Chapter 4 presents the method to generate sampling data.
- Chapter 5 introduces the concepts of the domain specific language and our proposal DLS.
- Chapter **??** we presents our framework with all components.
- Chapter **??** we discussed a case study performed with our solution.
- Chapter **??** we conclude our results.

CHAPTER 2

KEY-VALUE MODEL, MAPREDUCE AND HADOOP

This chapter introduces some concepts that are used in the subsequent sections: the key-value model, MapReduce paradigm and Hadoop framework.

2.1 Key-value model

Key-value model is a simplified model for data storage. It is based on one linked pair: the key and the value. Generally the pair is stored pure without aggregation or any creation of data schema, thus all detailing of data is done in runtime. Unlike other models such as relational model [7] in which the simplistic notion of relation already gives some sense for the data, similarly to the hierarchical data model [26] in which the links that connect the records give details for the data.

To solve some particularities involving relational model was created data warehouse, it is a repository that aggregates data from several sources [26], to make such aggregations is used the technique Extract Transform Load(**ETL**), the data are extracted from sources, so they are transformed and load in a data warehouse.

Due the simple storage of the data in key-value model the data *transformation* is done in the last fase, in other words, the data make sense when they are required, there is a inversion of ETL to ELT(Extract Load Transform), but such inversion cause one trouble to process a large amount of data, thus one big computing power is necessary to query the data. One programming paradigm that handles the key-value model is the MapReduce, it is going to present in the next section.

2.2 MapReduce

MapReduce is a programming system for high-level that allow many processes of one database can be written in simple way, according by Molina [19]. That database processes aim to process a large amount of data that are splitted and assigned to set computers, called computers cluster. Thus can improve the performance obtained by parallelism, omitting all complexity for that the user focus stays in the main problem that is the data processing.

The MapReduce paradigm have been implemented under key-value model, it was created to process a large amount of data and benefits from data parallelism, consequently builds large-scale parallel data processing applications. The paradigm is inspired on the high-level **Map** and **Reduce** primitives from functional programming languages. Hence the programmers can focus only in creation of the two higher-order funcions to solve a specific problem and to generate the necessary data, so it can just define the precice behavior of those functions.

Acording by[8]: "the computation takes a set of *input* key/value pairs, and produces a set of *output* key/value pairs.". The user write the map function that receives a set of input key/value pair and produces an *intermidiate* set of key/value pairs. The reduce function written for the user receives the intermidiate set as input and produces the *resultant* set of key/value pairs. This process is shown in Figure 2.1:

2.3 Hadoop

Map and reduce functions are present in Lisp and others functional languages. Recently the MapReduce paradigm have been implemented by several frameworks such as Greenplum MapReduce [14], Aster Data [2], Nokia Disco [22], Microsoft Dryad [18], among others. One open-source implemantation is the Hadoop that is a framework for reliable, scalable, distributed computing [17].

The Hadoop provide an interface to implement the map and reduce functions in high-level. It was projected for the user focus just on the implementation those functions,

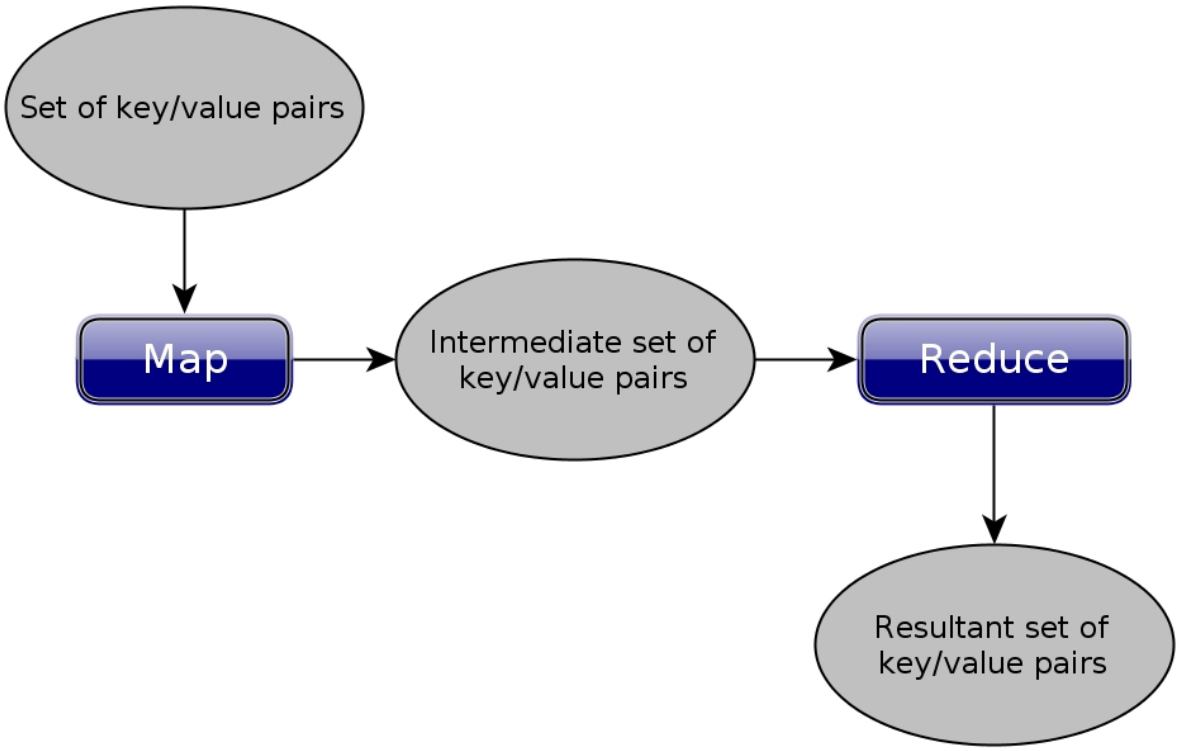

Figure 2.1: Map and Reduce process.

without worrying with the issues involving the distributed computing. All aspects involving the distributed computing and storage are left to the framework such as split files, replication, fault tolerance, tasks distribuition etc.

There are two main components on Hadoop:

- Hadoop Distributed File System(HDFS);
- Engine of MapReduce.

The HDFS stores all files in blocks, the block size is configurable per file, all blocks of one file have the same block size except the last block. It is divide in two components the *NameNode* and *DataNode*. The NameNode is placed in one master machine, it store all metedatas and manages all DataNodes, any aspect involving distributed storage is responsible by this component. The DataNode stores the data, when one DataNode starts it connects to NameNode, then responds to requests from the NameNode for filesystem operations.

The engine of MapReduce is responsible by the parallel processing, it is constituted

by one master machine and a lot of slave machines, also called workers. The master designates which slaves will receive map and reduce tasks with its respective input blocks. The worker who receive a map task is called mapper and the slave who receive reduce task is called reducer. All aspects involving the distributed computing management is responsible by the master like mappers failure, reducers failure, scheduling tasks, shuffling intermediate files etc.

2.3.1 Job processing

A job is a program in high-level languages(java, ruby or python) that implements the map and reduce functions. The master machine receive job submission with the relative input directory in the HDFS where are all files to process. This files must be inserted previously in the HDFS. Then the master requests to the NameNode infomation about the blocks and file locations, after that it deploys copies of the job across several workers.

With the blocks information acquired the map task is scheduled to a set of workers with its respective input blocks, then the mappers process each input blocks, generate key/value intermediate pairs and append its in intermediate files, when the mapper instance terminate it notify the master. The master splited the intermediate files in blocks and shuffled it to the reducers to process, when all reducers intances terminate, they append their result to the final output file. The data flow between mappers and reducers are shown in Figure 2.2.

The whole processing is based on $\langle key, value \rangle$ pairs. The mappers receive the file blocks, the mappers call the map function and pass the line number as key and the line as the value, so the pair "line number/line contente" is the $\langle k1, v1 \rangle$. The map generate the intermediate result set of key and values $\langle set(k2, v2) \rangle$, when the mappers finished all values for *k2* are agrouped in a list and the respective pair $\langle k2, list(v2) \rangle$ is generated. This pairs are sorted and pass as input for reducers that generate the result set:

map	$k1, v1$	$\rightarrow$	$set(k2, v2)$
reduce	$k2, list(v2)$	$\rightarrow$	$set(v2)$

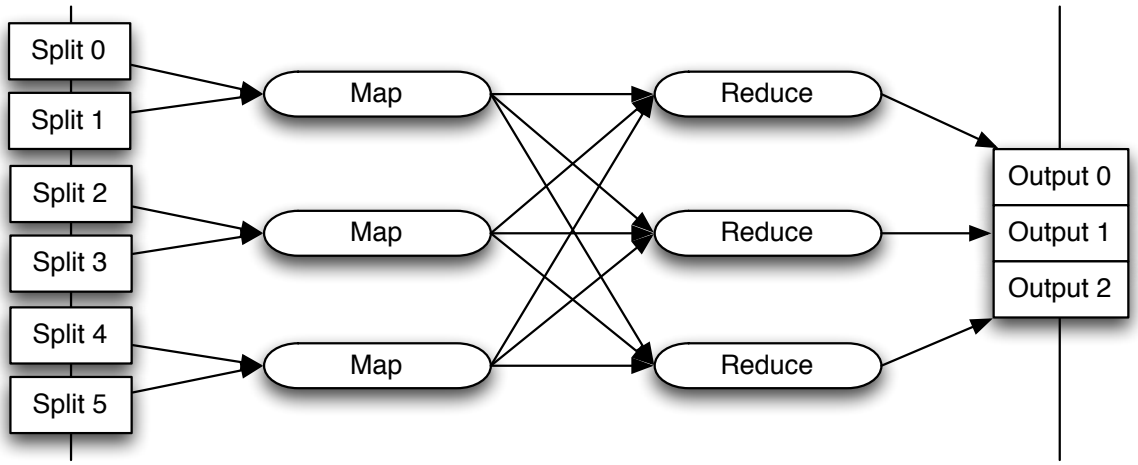

Figure 2.2: Execution of Map and Reduce operations

Eventually, when the map result are already available in memory, a local reduce function *Combiner* is used for optimization reasons, then all values for determinated key are combined, resulting in a local set $\langle k2, list(v2)\rangle$. This function runs after the Map and before the Reduce functions and is run on every node that run map functions. The Combiner may be seen as a *mini-reduce* function, which operates only on data generated by one machine.

A good example of a MapReduce job is the Grep application, which receives as an input several textual documents and as an output a set of pairs $\langle Key, Value\rangle$, where each key is a different pattern found and the value is the number of occurrences of the pattern in the files. The responsibility of the Mapper is to find pattern in the files and the reduce is to sum the amount found each patterns.

The Java implementation of the map function is presented in Listing 1. The **map()** method has four parameters: **key**, which is never used; **value**, one line that contains the text to be processed; the **output**, which will receive the output pairs and **reporter** for debug. The body of the method uses the class **Pattern** to describe a desired pattern, the class **Matcher** to find this pattern, when pattern are found the pair $\langle matching, 1\rangle$ is emited to output.

```java
public class RegexMapper<K> extends MapReduceBase
                        implements Mapper<K, Text, Text, LongWritable> {

    private Pattern pattern;
    private int group;

    public void configure(JobConf job)
    {
        pattern = Pattern.compile(job.get("mapred.mapper.regex"));
    }

    public void map(K key, Text value, OutputCollector<Text, LongWritable> output,
                                        Reporter reporter) throws IOException {
        String text = value.toString();
        Matcher matcher = pattern.matcher(text);
        while (matcher.find())
        {
            output.collect(new Text(matcher.group()), new LongWritable(1));
        }
    }
}
```

Listing 1: Class RegexMapper packed in Hadoop [17]

The implementation of the reduce function is presented in Listing 2. The **reduce()** method has also four parameters: **key**, which contains a single matching string; **values**, a set containing all values associated to the key (i.e. the matching); **output pair**, the resultant pair $\langle matching, total \rangle$ and **reporter** for debug. The behavior of the method is quite simple, it sums all values associated to the key and then writes a pair containing the same key and the total of matching found.

```java
public class LongSumReducer<K> extends MapReduceBase
                        implements Reducer<K, LongWritable, K, LongWritable> {

    public void reduce(K key, Iterator<LongWritable> values,
                       OutputCollector<K, LongWritable> output, Reporter reporter)
                throws IOException {

    // sum all values for this key
    long sum = 0;
    while (values.hasNext())
    {
        sum += values.next().get();
    }

    // output sum
    output.collect(key, new LongWritable(sum));
  }

}
```

Listing 2: Class LongSumReducer packed in Hadoop [17]

An example of the inputs and the outputs of both functions when applied to a simple sentence is presented in Table 2.1. We aplied the following regular expression:

”**[a-z]∗o[a-z]∗**”, this expression find the words that contains the vowel **o** in the midle of

them.

map	"Test for hadoop regular expression inside hadoop"	$\rightarrow$	$\langle for, 1\rangle$,$\langle hadoop, 1\rangle$, $\langle expression, 1\rangle$, $\langle hadoop, 1\rangle$
reduce	$\langle for, \{1\}\rangle$, $\langle hadoop, \{1, 1\}\rangle$, $\langle expression, \{1\}\rangle$	$\rightarrow$	$\langle for, 1\rangle$,$\langle hadoop, 2\rangle$, $\langle expression, 1\rangle$

Table 2.1: Regular expression example

CHAPTER 3

ALGORITHM FOR TEST

In this chapter we present the bacteriological algorithm, used to generate and select the job configurations on Hadoop.

3.1 Genetic Algorithm

Evolutionary Algorithms are the technique inspired on biological evolution process, it aims to select the best inviduals that adapt themselves in the environment. For this adptation is used biological mechanisms such as **reproduction**, **mutation**, **recombination or crossover** and **selection**. One the most known evolutionary algorithms is the Genetic Algorithm, it is closely related to evolution process.

The Genetic Algorithm work on the gene level, so all changes are done in this level. Although the gene seem to be one component without much relevance, but changes done its can be crucial for adptation in the environment, i.e., the genetic changes can be crucial to survival of one entire population of individuals or even mean survival of a species. Rosenzweig [25] cites that in the evolution process barriers may exist, like geographical barrier retricts gene flow within a sexually reproducing population and these genes could define the existence another population.

The Genetic Algorithm process describe in Figure 3.1 has its main strategy based in tree biological mechanisms: **reproduction**, **crossover** and **mutation** which are further detailed below:

- **Reproduction**: copies the individuals to participe of the next stage (the crossover), they are chosen as their abilities in adapt themselves in the environment, those abilities can be calculated according with a function *F(x)* that is called as the fitness of the individual like described in Figure 3.1. The copy ratio of the individual is based in your fitness, the choise is similar spinning a wheel where each invidual

receive slots according with your fitness. Thus the individual fitness is greater, then your number of copies tends to be greater.

- **Crossover**: the crossover is similar the natural process called chromosomal crossover, this process is basead on genetic recombination of chromosomes that produce new genetic combinations. Basically the genetic pair of two individuals are combined to generate another genetic pair to resultant individual, so the new individual has some characteristics of both parent. More minutely in the genetic algorithm two individuals are chosen randomly **(A, B)**, an integer k, between 0 and the size n of an individual less one, is chosen randomly. The new individual *A'* is composed by the first k genes of A and the last k - n genes of B. The individual *B'* consists of the first k genes of B and the last k - n genes of A.

- **Mutation**: after the crossover stage one mutation occur in the genes of new individuals. The process is simple in which one or more genes are selected randomly and then are changed (e.g.change one nucleotide of the DNA of one chromosome, or one gene is constituted for bits 0 or 1 and one bit is changed from 0 to 1).

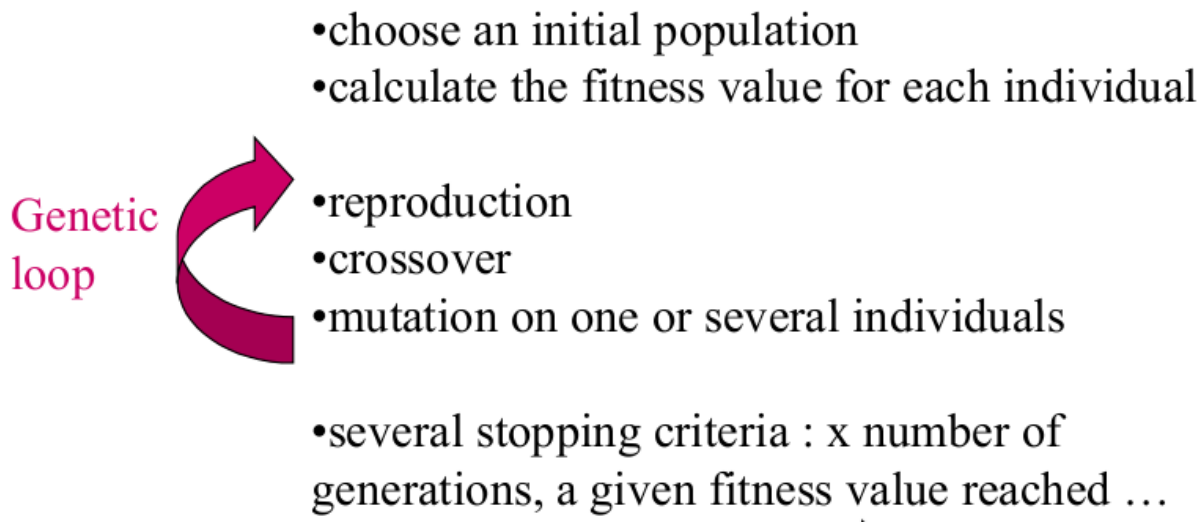

Figure 3.1: Genetic Algotithm process - Figure extracted of [3].

The algorithm begins with an initial population, for each individual is calculated your fitness that is the base for reproduction mechanism, so the three biological mechanisms is

called in the specific order already detailed and so the resultant population is evaluated as one or more criteria, if necessary the three mechanism are run again and the process continues until the criteria to be achieved.

3.2 Bacteriological Algorithm

Now in the Bacteriological Algorithm(BA) the individual is one bacteria and the focus of the algorithm is to adapt itself in a given specific environment. The algotithm is inspired on Genetic Algorithm(GA), but it have some peculiarities that improve some issues involving the GA and change your behavior.

The BA is more one adaptive approach of GA than one otimization, it introduces a new mechanism called memorization that is responsable to memorize the best individuals created along of the generations. As described in [3] it was proposed to improve the convergence of the GA, the introduction of the new mechanism might appear one small modification, but is actually reflect one crucial change of idea about GA.

Besides of the introduction the new mechanism, the old mechanism crossover was removed because of the bacteria behavior on its adaptation process in the environment. This mechanism cannot be used anymore, in terms of natural bacteriologic process the remotion of the crossover make sense, the bacteria reproduce themself asexually, the reproduction process consist in duplication of DNA and an after division to form two new cells.

The algorithm in high-level of abstraction is described in Figure 3.1, it is fed for one initial population of bacteria, after the bacteriological loop is started and has four main mechanisms: **Fitness computation**, **Memorization**, **Reproduction** and **Mutation** which are further detailed below:

- Fitness computation: the fitness analogously in GA is one way to differentiate the abilities of each individual in adapt themselves in the environment, its calculation depends of one or several criteria defined for the programer and it is used to select the best individuals for the next generation.

- Memorization: is the main mechanism introduced by the BA. Its is responsable for memorizing the best individuals generated in the process of adaptation, as the process continues the population improve more quickly our capacity of adaptation. The process consist in memorize the best individuals through of the generations, if one generation generate bad individuals i.e. generate low fitness values, then the memorization operator can ignore this generation and use the best individuals already generated in the past to the next generation, so avoid regressions in the process.

- Reproduction: is similar in GA, the best individuals are sorted ramdomly and they are selected to the mutation process. One important point can arise in this stage, the population size can grow up exponentially, so thresholds must be established.

- Mutation: this stage is responsible for generate new individuals, one or several genes are changed in order to improve the adaptation of the bacteria population in the environment. These new individuals are evaluated by their fitness values and they may or may don't be inserted in the set of best individuals.

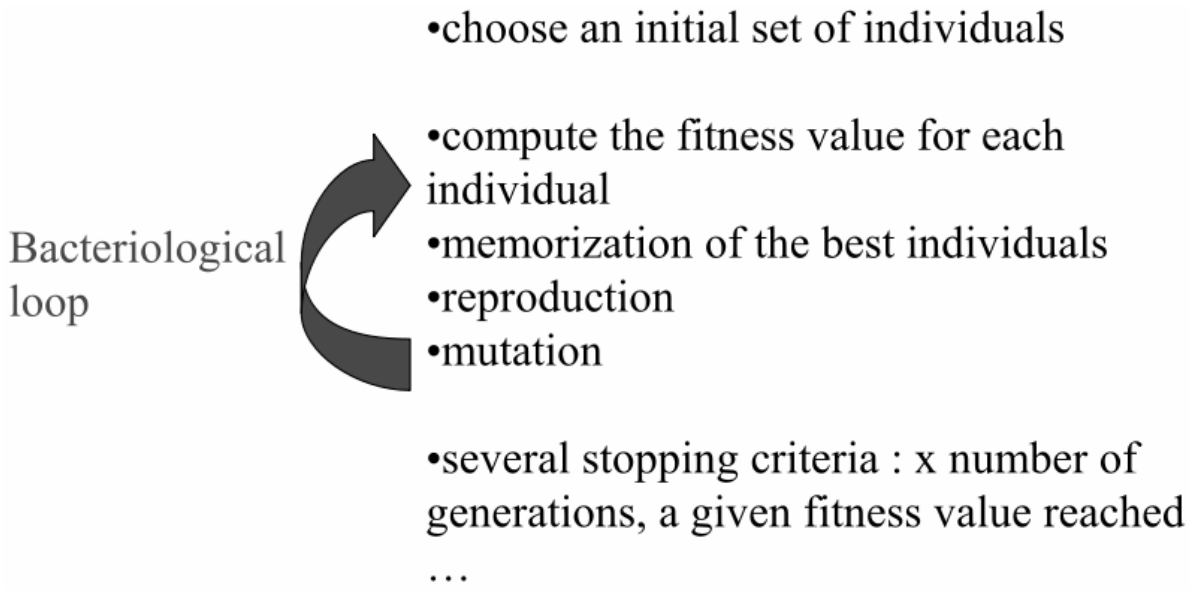

Figure 3.2: Bacteriological Algotithm process - Figure extracted of [3].

CHAPTER 4

SAMPLING ON HADOOP

In this chapter we present one method for data sampling on hadoop that is basead in a random algorithm.

4.1 Motivation for sampling

One relevant aspect in Big Data environment is to work with vast amounts of data, such fact is the main barrier to find a good configuration of one job. The bacteriological algorithm is a good option to create new configurations, but these configurations must be tested in order to select what is the best for the job in question at the moment. Another relevant issue in this type of environment is your volatility, caused by constant variation of the data, i.e. insertion or remotion of data constantly. Also due processing the large amounts of data the power computing depends of hundreds or even thousands of machines and it may fail, this fails characterize changes on the environment what can invalidate the current configuration for the job.

Therefore, there is one cycle on the Big Data environment that is to select one configuration for a given job and execute the job, and again chosen one configuration and execute the job etc. Such cycle must repeat on the environment due your feature of high volatility.

One issue stay in the air: *how must we test and select the job configurations generated by bacteriological algorithm?* One possible answer is to run the job with sample data, because the job wouldn't run on all data stored what would spend too much time and also would be impracticable due larger number of intermediate configurations generated by the algorithm.

4.2 Challenge for sampling in Big Data environment

On Big Data environment there are several aspects involving computing and storage distributed. For data sample the aspects involving storage distributed are more relevant. In this context, without doubt, the data volume is the main issue because the data sampling must be done distributed too, otherwise one machine couldn't bear all data storage in the cluster then make the data sampling. Also the data sampling resultant must be storage distributed in the cluster, because even being a sample the result can be big and a single machine couldn't bear too.

So the sampling must be done distributed and it must be done without intrusion in the Hadoop, because any change done inside of the framework can propagate collateral effects due its complexity. Further more to run test regression is a very costly activity [23] and to create unit test cases for the new changes spend much time.

One way to sample data on Hadoop is to utilize its benefits, i.e. to benefit of its structure of storage and computing distributed. We can build one MapReduce program to sample data and so we will be benefiting of its advantages as framework to distributed computing.

One of the most used data sample techniques is **Random Sampling** that consists basically in select a pre-determined amount or percentage of data randomly [24]. In the literature there are several others techniques such as **Stratified Random Sampling**, this techniques splits the data in strata in which each element has the same chance of being selected [24]. Another thechnique is **Systematic Sampling** that chooses randomly the first element and then till the k-esimo element in sequence is selected [13].

In the context of big data there are some implementations of data sample. One example is the **MonetDB** which is column-oriented database management system and was designed to hold data in main-memory and processes large-scale data distributed [21], this database support data sample and use the *Algorithm A* that is based in random sample method [28].

Another database management system that performs data sample based in random method is the **Hive** that is data warehouse system for Hadoop. He done sampling in row

or block size level. The row level consist in choose randomly the rows according with colunm name, if the column name is not defined then the entire row is selected, with the colunm name defined the choice can be done using **Bucketized Table** which was hash-clustered by columns [1], so the sample is done only on the buckets that contains the specified column. The block size sample is done ramdomly too and consist in to select the blocks that match with the specified block size.

Those sample methods on Hive are based in random sample and handle structured data. The Hive principle is just store the Hadoop data as a data warehouse and facilitate queries submitted by users. Moreover, the clustering by bucket and block size concept requires a prior structuring of data, so in the Hive several information about the data are previously known.

In the Hadoop the data are stored unstructured and this characteristic is the biggest trouble to develop data sample on Hadoop. According with [27, 6, 30] the trouble with unstructured data stream can be solved with **Reservoir Sampling**, it consist in solved this issue: "*Say you have a stream of items of large and unknown length that we can only iterate over once. Create an algorithm that randomly chooses an item from this stream such that each item is equally likely to be selected.*"

The Reservoir Sampling is part of the randomized algorithm family and consist in choose randomly k elements from a list L containing N items. Normally the length N is either unknown or large enough that the memory doesn't support such list. The algorithm is shown below in pseudocode:

The basic idea of this algorithm is build a reservior smaller than the memory. So it receive as parameter the number k that is the resultant sampling length and *stream* of data that constantly receive new data. Initially, the resultant *arraySample* is assigned with the first k items of stream, after until the end of stream is sorted randomly one number *rand* between 1 and current length of data already received, if the random number is smaller or equal than k then the *arraySample* position *rand* is assigned to new data arrived.

However, choose the number k is a hard task because the resultant sample must be representative, besides the primary memory may not support the array resultant. To solve

Algorithm 1: Algorithm for Reservoir Sampling

```
Input  : k size of sample
Input  : stream data stream with indefided length
Output: arraySample[k]
for i = 1 → k do
    arraySample[i] ← stream[i]
currentLength ← k
while stream != EOF do
    currentLength ← currentLength + 1
    rand ← Random(1, currentLength)
    if rand ≤ k then
        arraySample[rand] ← stream[currentLength]
return arraySample
```

that problem Vitter [27] suggests that the reservoir be stored on secundary storage, but this approach is implactible on Hadoop context, because the secundary storage is HDFS, so the time of retrieving the reservoir and update is very large.

4.3 One method for data sampling on Hadoop

We present one algorithm based on MapReduce paradgm to generate data sample on Hadoop. So we will be taking advantage the structure of the framework, consequently all aspects related with storage and computing distributed is left for it.

Our idea consist in build one MapReduce program using randomized data sampling. As we have seen the random algorithms for data sampling have been used largely in database and big data environment. The challenge here is to generate a representative sample that is also one of the biggest challenges in statistic area.

First of all, we will present the behavior of the algorithm on map and reduce process using the Figure 4.1, the process receive tree files as example file1, file2 and file3, there are two mappers who classified each line of the files and generate the output $\langle lineNumer, content\rangle$, after the shuffle step agregates each content of the same key $\langle lineNumer, \{content1, content2, ..., contentN\}\rangle$, then the only reducer receives the shuffle output, for each content of the same key chooses one random number and if it is less than sample threshold then chooses this content, otherwise discards it.

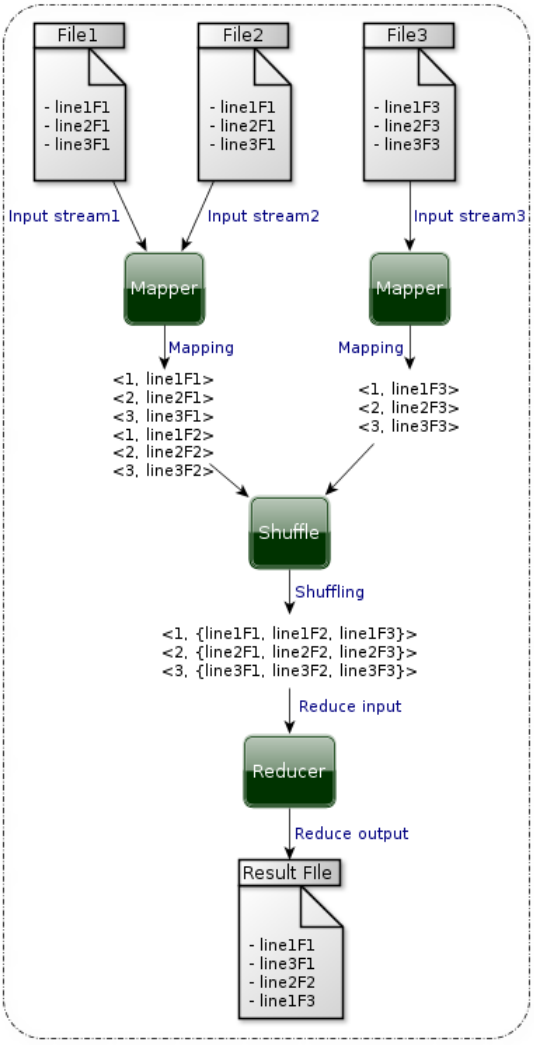

Figure 4.1: Map and Reduce random sample process

With the process behavior already shown is easier explained the algorithm, it was splited in map and reduce function:

The map function consist basically in classify each line in the file stream with it respective order of processing, then emits the intermediate key-value pair $\langle lineNumer, lineContent \rangle$, each pair is added in a map structure that is the output of the map function. After the shuffle phase aggregates values that share the same key. This aggregated is the input for

Algorithm 2: Map function for data sample

```
Input  : fileStream stream of file lines
Output: map < Integer, String > list of key-value
lineNumber ← 1
map ← {}
for each line ∈ fileStream do
    line ← fileStream.getNextLine()
    map.put(lineNumber, line)
    lineNumber++
return map
```

reduce function:

Algorithm 3: Reduce function for data sample

```
Input  : mapList < Integer, List < String >> list of key-value aggregated by
         shuffle phase
Output: list of selected lines
list ← {}
for each key ∈ mapList do
    length ← mapList.get(key).length()
    for each line ∈ key do
        rand ← random(0, length − 1)
        if rand < length then
            list.add(line)
return list
```

The reduce function performs the sampling strictly speaking. First it iterates in each key belonging of the input list, each line that share the same key, i.e. the same order of processing, it chooses one random number between the 0 and the amount of line for the same key at least one, if this random number is less than amount of lines then the value is added to resultant list. So in the end of process we obtain a data sample.

CHAPTER 5

DOMAIN-SPECIFIC LANGUAGE

In this chapter we present some fundamentals of Domain-Specific Language and one language in our domain that will be helpful for user utilize our framework.

A *domain-specific language* (DSL) is way to approach of some specific context through appropriate notations and abstractions [11]. DSL transforms a particular problem domain into a context intelligible for expert users that can work in a familiar environment.

Problem domain is a crucial term of DSL that requires prior background of the developers in the specific context, so the developers must be expert in the domain in order to develop DSLs that cover all features required for the users. There are a lot of examples of DSLs in differents domains, **(LEX, YACC, Make, SQL, HTML, CSS, LATEX, etc.)** are classical examples of DSLs [4].

DSLs are usually focused in its domains containg notations and specific abstractions, normally DSLs are *small* and *declarative* languages. However, a DSL can be extended to others domains, in this case such DSL is general-purpose language (GPL), because its expressive power is not restricts an exclusive domain, examples of such DSLs are **Cobol and Fortran**, which could be viewed as languages focused towards the domain of business and scientific programming [11], respectively, but they are not restricts just in this domains.

DSL are used in several big areas, such **Software Engineering**, **Artificial Intelligence**, **Computers Architecture**(in this area a good exemple is VHSIC Hardware Description Language (VHDL), where VHSIC mean **V**ery **H**igh **S**peed **I**ntegrated **C**ircuits), **Database Systems**(SQL is a classical example already cited), **Network**(where its protocols are examples of DSLs), **Distribuited Systems**, **Multi-Media** and among others. A current area that have been emerged recently is **Big Data**, this area may be considered as a sub area of Database, but is has many particularities that involve a mix features of

Database and Distributed Systems.

5.1 DSL Design Methodology

The first step to create a new DSL is identify the problem domain. Depending on context is not so easy to identify the domain, can there are many particularities involving the full understanding of the domain, also the context can cover more than one domain, for example the GPLs. In other cases the correct identification of the domain is fast and there is not margin for doubts and equivocation. In both cases the foreknowledge of the developers is the factor that more influences in good or bad DSLs resultants.

After to identify the problem domain the developer must abstract all relevant aspects in this domain. This is an important phase in building of any software, similar the phase of definition of the business rules, when architects and developers decide what aspects are relevants or not for the users. Such decisions are going to reflect in the usability and somehow acceptance of the new software.

With all relevant aspects well defined the knowledge acquired can be clustered in a set of semantic notations and operations on them, that set is related with the expression power of the language. This group of semantic notations and operations someway will be available for the users.

So the next step is design a DSL that expresses applications in the domain, the new DSL will have limited concepts which are all focused on specific domain. For design the DLS is necessary to analyse the relationship between it and the existing languages. According with [20] there are some design patterns to develop a DSL based in existing languages that is represented by figure 5.1.

In the implementation is constructed a library with the semantic notations together with a compiler that perfoms the lexical, syntactic and semantic analysis, after converts the DSL programs to sequence of library calls. Generally the library and the compiler are constructed with support of the tools or framework developed for this purpose. **Xtext** [12] and **Groovy** [15, 10] are good examples of tools to develop DSLs quickly.

Pattern	Description
Language exploitation	DSL uses (part of) existing GPL or DSL. Important subpatterns: • Piggyback: Existing language is partially used • Specialization: Existing language is restricted • Extension: Existing language is extended
Language invention	A DSL is designed from scratch with no commonality with existing languages
Informal	DSL is described informally
Formal	DSL is described formally using an existing semantics definition method such as attribute grammars, rewrite rules, or abstract state machines

Figure 5.1: Design patterns - Figure extracted of [20].

5.2 Context Transformation

The bacteriological algorithm is part of the family of genetic algorithms that works on genetic context. A minor particle this algorithm is a gene, but without minor importance, all changes are influenced by it. The genes when clustered form an individual that have more representativeness than an gene and the top is the population that is set of individuals.

Our context is focused on hadoop environment that have your particularities. Thus a context transformation is mandatory to implement the bacterionlogical algorithm on such environment. This is not a complicated task for who know the hadoop environment that is our specific domain.

On hadoop there is huge set of configuration parameters, we called one specific parameters of **knob**, but a job use several knobs that are one knobs set. When some knobs set are gathered we have a population of knobs.

So pure and simples tranformation have been done where each component of genetic context were translated to one component of hadoop environment. Like shown in the figure 5.2 we can realize that one gene was transformed to one knob, one individual(that is a genes set) was transformed to one knobs set and one individuals population was transformed to knobs population.

An interesting characteristic of the tranformation is its bijection that one component in genetic domain is translated to one component in hadoop domain. Beyond that the

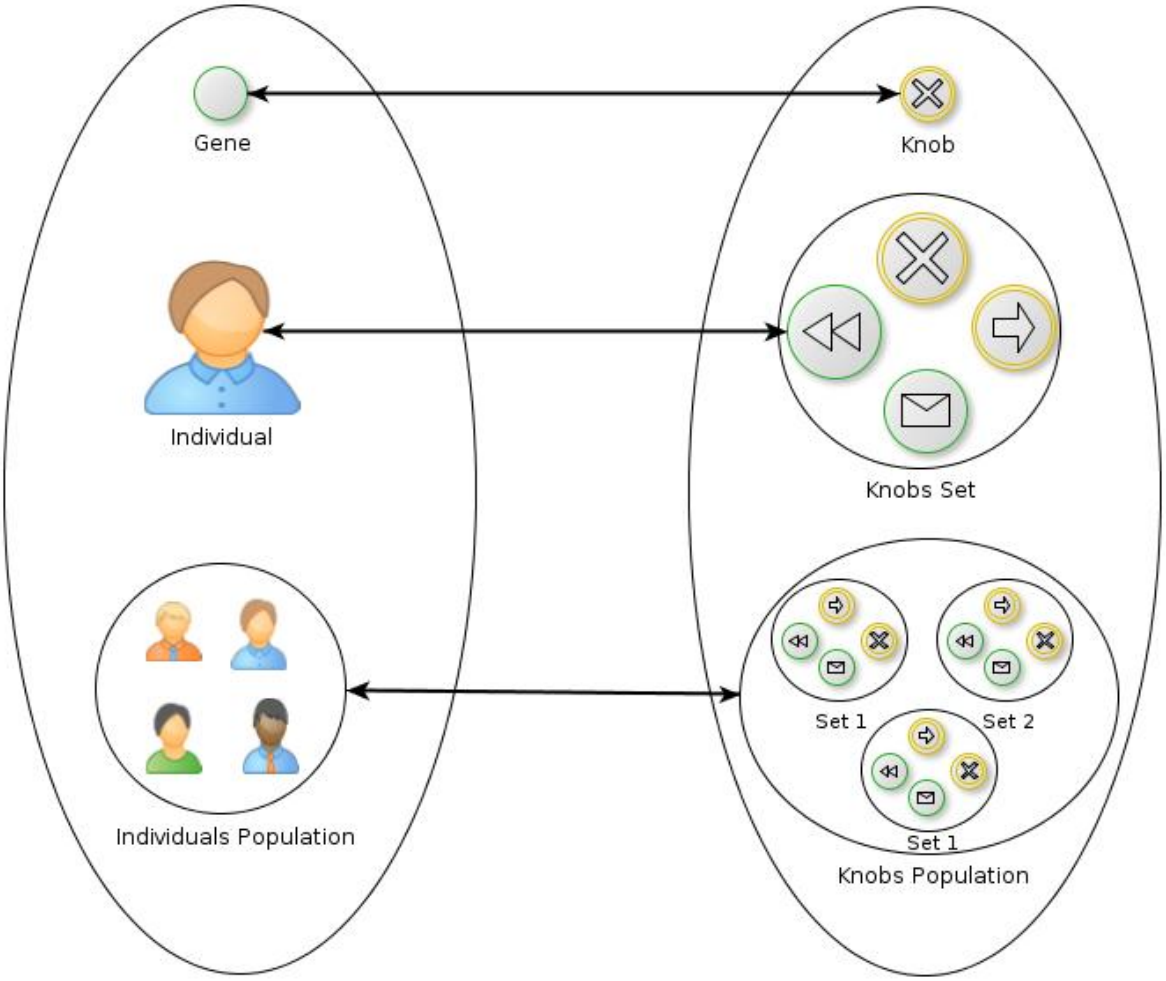

Figure 5.2: Context transformation.

transformation has inversion property, i.e, all components in hadoop domain can be translated to respective components in genetic domain. That properties represent compatibility between both domains and somewhat a good representativeness.

5.3 DSL Proposal

The base of our DSL proposal is context transformation already explained, but to develop the DLS still missing one library or framework that help us in this task. The **Xtext** [12] framework seems one good option because is simple, covers all our necessities and we already have knowledge with this tool.

Our effort concentrate in one MapRedue job and your knobs to be adjusted. One draft of DLS is shown above:

```
DomainModel:
        job=Job;

Job:
        'Job' name=ID '{'
                setKnobs+=Knobs*
        '}'
;

Knobs:
        'knobs' '{'
                knobs+=Knob*
        '}'
;

Knob:
         name=ID Type
;

Type:
        IntType | FloatType | BoolType
;

IntType:
        'int' MinInt MaxInt '=' INT
;
MaxInt: INT;
MinInt: INT;

FloatType:
        'float' MinFloat MaxFloat '=' Float
;
MaxFloat: Float;
MinFloat: Float;

Float:
        INT*'.'INT*
;

BoolType:
        'boolean' '=' Boolean
;
Boolean:
        'true' | 'false'
;
```

Listing 3: Initial DSL proposal

Let's explain all rules involving our grammar:

1.
```
DomainModel:
        job=Job;
```

The first rule in a grammar is always used as the entry or start rule. It says that the **DomainModel** contains one element **Job** assigned to a feature called *job*.

2.
```
Job:
        'Job' name=ID '{'
                setKnobs+=Knobs*
        '}'
;
```

The rule **Job** starts with the definition of a keyword (*Job*) followed by a name. Between 'braces' the job contains one arbitrary number (*) of **Knobs** which will be added (+=) to a feature called setKnobs.

3.
```
Knobs:
        'knobs' '{'
                knobs+=Knob*
        '}'
;
```

The rule **Knobs** starts with the definition of a keyword **knobs** and between 'braces' contains one arbitrary number (*) of **Knob** which will be added (+=) to a feature called knobs.

4.
```
Knob:
         name=ID Type
;
```

The rule **Knob** contain one name followed by a **Type** with your peculiarities explained below.

5.
```
Type:
        IntType | FloatType | BoolType
;
```

The rule **Type** can accept three type: integer, float or boolean, this three are all possibles types on hadoop parameters configuration.

6.
```
IntType:
        'int' MinInt MaxInt '=' INT
;
MaxInt: INT;
MinInt: INT;
```

This three rules are used for integer types, the rule **IntType** starts with the keyword **int** followed by your respective minimum and maximum possibles values. In sequence there is the keyword = and the initial value for the knob.

7.
```
FloatType:
        'float' MinFloat MaxFloat '=' Float
;
MaxFloat: Float;
MinFloat: Float;

Float:
        INT*'.'INT*
;
```

This four rules are used for float types, the rule **FloatType** is similar the IntType rule, it starts with the keyword **float** followed by your respective minimum and maximum possibles values. In sequence there is the key word = and the initial float value for the knob. The rule **FloatType** expresses the float format.

8.
```
BoolType:
        'boolean' '=' Boolean
;
Boolean:
        'true' | 'false'
;
```

The last one rule **BoolType** expresses the boolean type, it starts with the keyword **boolean** followed by signal of = and the initial boolean value that can be **true** or **false**.

CHAPTER 6

THE FRAMEWORK

In this chapter we present our framework that composite by three modules: one front-end for the users iteract with the system, one engine to choose good job configurations called tuning-by-testing and one back-end to report the new job configuration.

6.1 Framework front-end

The front-end is the *DSL* shown in the chapter 5, it consist in say what is the job name to predict a good configuration and one or several set of knobs with their initial values, which can be assigned with any values since the last configuration or rule of thumbs until a random assignment.

One use case of the DSL is shown below, it contains the job name **WordCount** and your set of initial knobs. This set forms the initial population for bacteriological algorithm, each knob set (**knobs**) represents one individual and one single knob corresponds one gene of individual, any change is done on gene level i.e. on knob level.

```
Job WordCount {
        knobs {
                "dfs.block.size" int 1024 1048576 = 4096
                "io.sort.spill.percent" float 0.0 1.0 = 0.5
                "mapred.map.tasks.speculative.execution" boolean = false
        }
        knobs {
                "dfs.block.size" int 1024 1048576 = 1024
                "io.sort.spill.percent" float 0.0 1.0 = 0.08
                "mapred.map.tasks.speculative.execution" boolean = false
        }
        knobs {
                "dfs.block.size" int 1024 1048576 = 1048576
                "io.sort.spill.percent" float 0.0 1.0 = 1.0
                "mapred.map.tasks.speculative.execution" boolean = true
        }
}
```

Listing 4: Usage of DSL Proposal

As the example shown there are three initial set of knobs to test, in this example the first set could be the last job configuration, the second the rule of thumbs suggested by the comunity and the last one random assignment. However, the users can interested in testing new knobs that would be easy, just only put a new set of knobs to test, so this front-end covers enough use cases or combination cases as much as the users wish.

6.2 Framework engine

The framework engine is divided in three components: the component to generate data sample, the component to auto configure hadoop with all configurations generated by the core component and the last one is core component to choose job configurations that was implemented using BA.

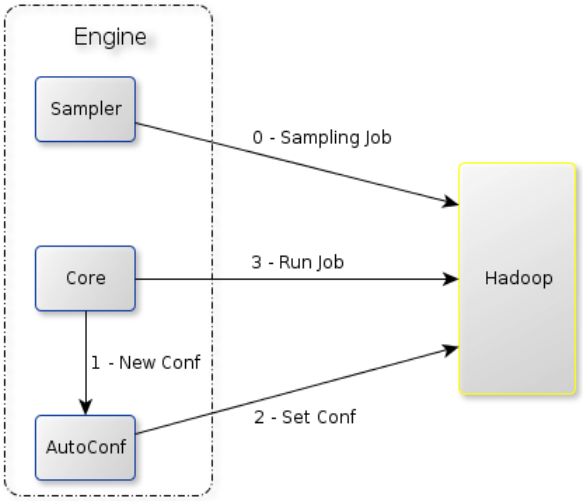

Figure 6.1: Engine processing.

6.2.1 Sampler component

engine

The sampler is responsable to generate data sample, so it sends a command to hadoop in order to run the sampling job and your output will be used by the core component, this step is represented by command **0 - Sampling Job**.

6.2.2 AutoConf component

The AutoConf is one component developed by Ramiro, E. colocar referência do auto-conf it is responsable for communicate with the hadoop and inject new job configuration, as seen in the command **2 - Set Conf**. So independent of configuration files or configurations assigned for the own job, the hadoop will use job configuration sent for AutoConf.

6.2.3 Core component

The core component chooses job configurations in order to test its performance and evaluate the latency time using the bacteriological algorithm, it sends the command **1 - New Conf** to component AutoConf after assigned the configuration, the job is submited to hadoop through the command **Run Job**, then your latency time is evaluated and the job configuration may added or not to list of good configurations. The commands sequence *1, 2 and 3* occur until the BA finish, i.e., until one criteria is reached.

6.2.4 Framework back-end

The back-end at the moment is undefined, currently the result is being saved in one file with the same format of the input file, but with an exception it contains just the best job configuration found by the BA until the criteria was reached.

6.3 Overview

In the Figure 6.2 we show all components together. Fist of all the user create one file containing initial knobs, this file is submitted to component front-end which performs lexical, syntatic and semantic analysis on the file, after it is parsed and sent to engine component. This component, how already explained, active the BA to choose one good job configuration until reach the criteria. The result is passed to back-end component that saved in a file.

One interesting feature is the result file can be used as input for the next round of the framework, just only the user submit it as input. So the framework can work as incremental software to improve its last result.

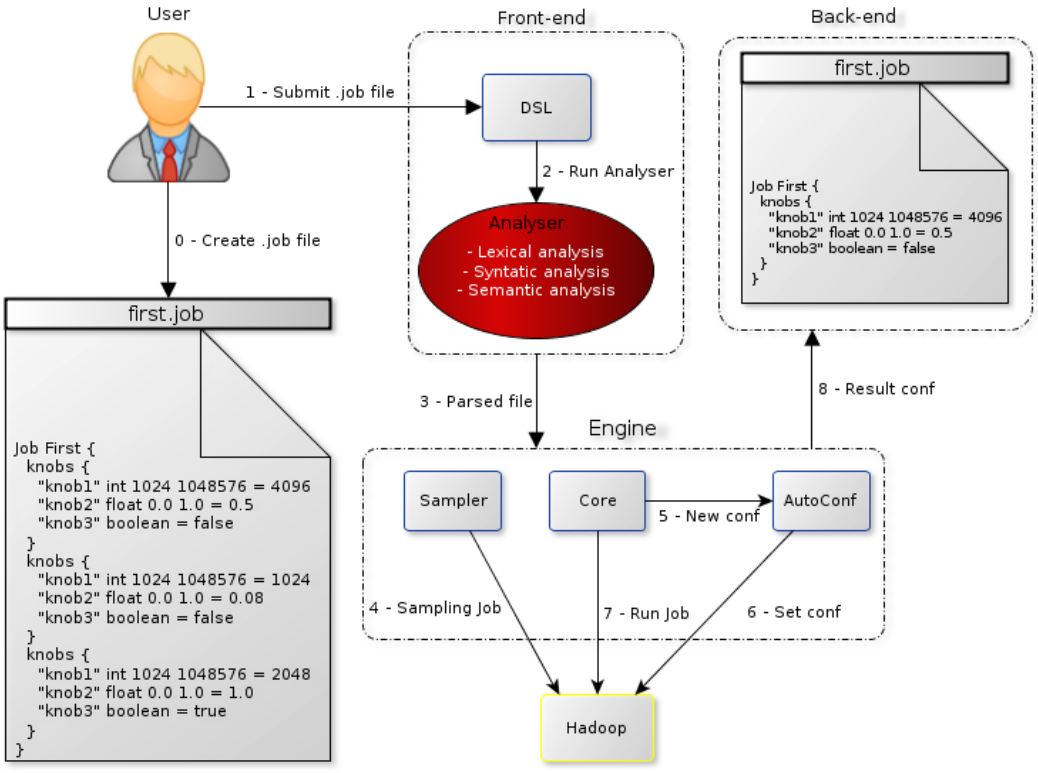

Figure 6.2: Framework overview.

BIBLIOGRAPHY

[1] apache.org. Languagemanual sampling. Site: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling, 2013. Accessed on 11th July 2013.

[2] Inc. Aster Data Systems. In-database mapreduce for rich analytics.

[3] Benoit Baudry, Franck Fleurey, Jean-Marc Jézéquel, e Yves Le Traon. From genetic to bacteriological algorithms for mutation-based testing: Research articles. *Softw. Test. Verif. Reliab.*, 15:73–96, June de 2005.

[4] J. L. Bentley. Programming pearls: Little languages. *Communications of the ACM*, 29(1):711–721, August de 1986.

[5] Yanpei Chen, Sara Alspaugh, e Randy Katz. Interactive analytical processing in big data systems: a cross-industry study of mapreduce workloads. *Proc. VLDB Endow.*, 5(12):1802–1813, agosto de 2012.

[6] cloudera.com. Algorithms every data scientist should know: Reservoir sampling. Site: http://blog.cloudera.com/blog/2013/04/hadoop-stratified-randosampling-algorithm, 2013. Accessed on 14th July 2013.

[7] Edgar Frank Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[8] Jeffrey Dean e Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[9] Jeffrey Dean, Sanjay Ghemawat, e Google Inc. Mapreduce: simplified data processing on large clusters. *In OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*. USENIX Association, 2004.

[10] Fergal Dearle. *Groovy for Domain-Specific Languages*. june de 2010.

[11] Arie Van Deursen, Paul Klint, e Joost Visser. Domain-specific languages: An annotated bibliography. *ACM SIGPLAN NOTICES*, 35:26–36, 2000.

[12] eclipse.org. Xtext. Site: http://www.eclipse.org/Xtext, 2013. Accessed on 1st April 2013.

[13] en.wikipedia.org. Systematic sampling. Site: http://en.wikipedia.org/wiki/Systematic-sampling, 2013. Accessed on 2nd July 2013.

[14] Greenplum. A unified engine for rdbms and mapreduce, 2008.

[15] groovy.codehaus.org. Domain-specific languages with groovy. Site: http://groovy.codehaus.org/Writing+Domain-Specific+Languages, 2013. Accessed on 1st April 2013.

[16] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin and S. Babu. Starfish: A self-tuning system for big data analytics. *In Proc. of the 5th Conference on Innovative Data Systems Research (CIDR '11)*, January de 2011.

[17] hadoop.apache.org. Apache hadoop. Site: http://hadoop.apache.org/, 2013. Accessed on 24th March 2013.

[18] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, e Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *EuroSys*, 2007.

[19] Jennifer Widom Jeffrey D. Ullman, Hector Garcia-Molina. *Database systems - the complete book*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2009.

[20] Marjan Mernik, Jan Heering, e Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, 37(4):316–344, 2005.

[21] monetdb.org. Monetdb. Site: http://www.monetdb.org, 2013. Accessed on 2nd July 2013.

[22] Prashanth Mundkur, Ville Tuulos, e Jared Flatow. Disco: a computing platform for large-scale data analytics. *Proceedings of the 10th ACM SIGPLAN workshop on Erlang*, Erlang '11, New York, NY, USA, 2011. ACM.

[23] Tauhida Parveen, Scott R. Tilley, Nigel Daley, e Pedro Morales. Towards a distributed execution framework for junit test cases. *ICSM*, 2009.

[24] Randomsampling.org. Random sampling. Site: http://www.randomsampling.org, 2013. Accessed on 2nd July 2013.

[25] Michael L. Rosenzweig. *Species Diversity in Space and Time*. Cambridge University Press, maio de 1995.

[26] Abraham Silberschatz, Henry F. Korth, e S. Sudarshan. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005.

[27] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[28] Jeffrey Scott Vitter. Faster methods for random sampling. *Commun. ACM*, 27(7):703–718, 1984.

[29] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, 1 edition, july de 2009.

[30] wikipedia.org. Reservoir sampling. Site: http://en.wikipedia.org/wiki/Reservoir-sampling, 2013. Accessed on 14th July 2013.

TIAGO RODRIGO KEPE

SELF-TUNING BASED ON DATA SAMPLING

Dissertation presented as partial requisite to obtain the Master's degree. M.Sc. program in Informatics, Federal University of Paraná. Advisor: Prof. Dr. Eduardo C. de Almeida

CURITIBA

2013