Distributed computing

Weights in a fertilizer factory

Rui S. Cruz
Dept. of Computer Science and Engineering, IST
University of Lisbon, and INESC-ID
Lisbon, Portugal
rui.s.cruz@tecnico.ulisboa.pt
ORCID iD: 0000-0002-4683-417X

Miguel Casquilho
Dept. of Chemical Engineering, IST
University of Lisbon, and CERENA
Lisbon, Portugal
mcasquilho@tecnico.ulisboa.pt
ORCID iD: 0000-0003-2062-370X

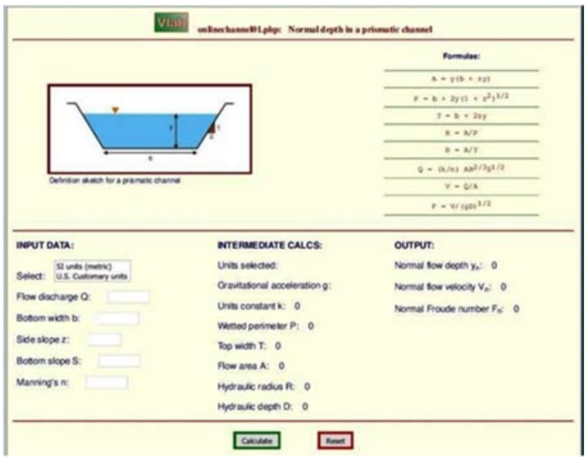
Abstract — **This study is based on a concrete problem in a fertilizer factory about the estimation of process parameters: to calculate the mean and standard deviation from weights (sums only) of loads of unequal (known) number of bags ("equal" case being trivial). With many distribution depots, the data for each depot must be collected for processing. These are addressed in a Cloud Computing, big-data framework. The use of Apache Spark is described and adopted, as advantageous over Hadoop due to "in-memory computation" and Resilient Distributed Dataset. The computation uses Terraform and Ansible as configuration tool, and is deployed on the Google Cloud Platform. The evaluation preliminary tests confirmed good accuracy and produced low runtimes.**

Keywords – estimation, distributed computing, Cloud, Apache Spark.

I. INTRODUCTION

The present study originated in a concrete question in industry, in a large fertilizer plant. The last operation in the factory, before delivery to the customer, is to fill the bags with the (granular) fertilizers with a "correct" weight (e.g., 25 kg). Our objective is to take advantage of the weighing of the (collective) loads of bags upon delivery in order to estimate their (individual) statistical properties: mean (μ) and standard deviation (σ) of the weight of the bag. Exactly the same problem exists in innumerable other industrial activities, such as transportation of components for manufacturing (e.g., bolts) or materials of construction (bricks), bagging fruits, packaging cakes, eggs, all these applications considering sets of variable but known cardinality, i.e., samples of unequal sizes. These sizes, though unequal, are known. The problem, which was mentioned by us in a previous edition of CISTI [1], is now the basis for a new perspective, an information technology view of distributed computing, supposing that there are more than one sources of data, as also happens in companies with several subsidiaries.

The base problem addressed in this study (a scenario of one factory or one depot) is soluble in a Web page of ours [2], as we always do in our daily scientific activity for this problem and many others (in fields related to Engineering, Operations Research). In our style of offering computing, the user needs no software installation or add-on, such as Java, javascript, etc., so any terminal able to browse the Internet can use it, irrespective of its power or operating system. In this style, the only other website we know of is Ponce's [3] outstanding "Vlab" [4], dealing with hundreds of problems in his field (Hydraulics), one example being shown in Figure 1.


Vlab
onlinechannel01.php: Normal depth in a prismatic channel
Formulas:
Definition sketch for a prismatic channel
Calculate
Reset

INPUT DATA:

Select: SI units (metric) / U.S. Customary units
Flow discharge Q:
Bottom width b:
Side slope z:
Bottom slope S:
Manning's n:

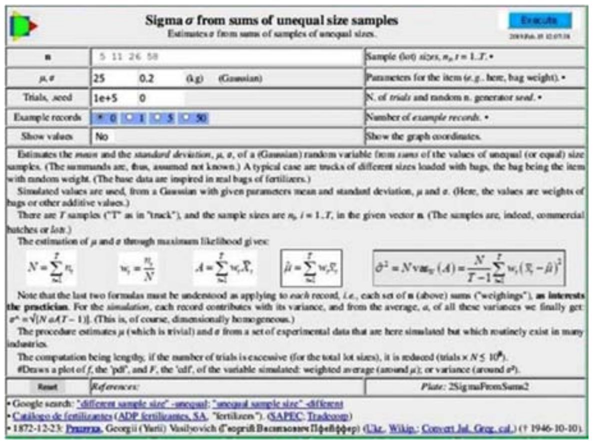
INTERMEDIATE CALCS:

Units selected:
Gravitational acceleration g:
Units constant k: 0
Wetted perimeter P: 0
Top width T: 0
Flow area A: 0
Hydraulic radius R: 0
Hydraulic depth D: 0

OUTPUT:

Normal flow depth y_n: 0
Normal flow velocity V_n: 0
Normal Froude number F_n: 0

Figure 1. Ponce's Problem "Normal depth in a prismatic channel using the Manning equation".



Sigma σ from sums of unequal size samples
Estimates σ from sums of samples of unequal sizes.

n	5 11 26 58	Sample (lot) sizes, n_i, i = 1..*T*. •
μ, σ	25 0.2 (kg) (Gaussian)	Parameters for the item (*e.g.*, here, bag weight). •
Trials, seed	1e+5 0	N. of *trials* and random n. generator *seed*. •
Example records	[✓ 0] 1 5 50	Number of *example records*. •
Show values	No	Show the graph coordinates.

Estimates the *mean* and the *standard deviation*, μ, σ, of a (Gaussian) random variable from *sums* of the values of unequal (or equal) size samples. (The summands are, thus, assumed not known.) A typical case are trucks of different sizes loaded with bags, the bag being the item with random weight. (The base data are inspired in real bags of fertilizers.)

Simulated values are used, from a Gaussian with given parameters mean and standard deviation, μ and σ. (Here, the values are weights of bags or other additive values.)

There are *T* samples ("T" as in "truck"), and the sample sizes are n_i, i = 1..*T*, in the given vector **n**. (The samples are, indeed, commercial batches or *lots*.)

The estimation of μ and σ through maximum likelihood gives:

$$N=\sum_{i=1}^{T} n_i \qquad w_i=\frac{n_i}{N} \qquad A=\sum_{i=1}^{T} w_i \bar{X}_i \qquad \boxed{\hat{\mu}=\sum_{i=1}^{T} w_i \bar{X}_i} \qquad \boxed{\hat{\sigma}^2 = N \operatorname{var}_W(A) = \frac{N}{T-1}\sum_{i=1}^{T} w_i \left(\bar{X}_i-\hat{\mu}\right)^2}$$

Note that the last two formulas must be understood as applying to *each record*, *i.e.*, each set of **n** (above) sums ("weighings"), **as interests the practician**. For the *simulation*, each record contributes with its variance, and from the average, *a*, of all these variances we finally get $\sigma^\wedge = \sqrt{[N\, a/(T-1)]}$. (This is, of course, dimensionally homogeneous.)

The procedure estimates μ (which is trivial) and σ from a set of experimental data that are here simulated but which routinely exist in many industries.

The computation being lengthy, if the number of trials is excessive (for the total lot sizes), it is reduced (trials × $N \le 10^8$).

#Draws a plot of *f*, the 'pdf', and *F*, the 'cdf', of the variable simulated: weighted average (around μ); or variance (around σ^2).

Reset	*References:*	*Plate:* 2SigmaFromSums2

• Google search: "different sample size" -unequal; "unequal sample size" -different
• Catálogo de fertilizantes (ADP fertilizantes, SA, "fertilizers"). (SAPEC: Tradecorp)
• 1872-12-23: [illegible], Georgii (Yurii) Vasilyovich (Георгій Васильович Пфейффер) (Ukr., Wikip.; Convert Jul. Greg. cal.) († 1946-10-10).

Figure 2. Our Web page for a single depot.

In our Web page for the problem under consideration, shown in Figure 2, in the single depot version, the data are simulated (Monte Carlo) at the user's will, producing data similar to the real ones.

The current study is a generalization of the base study, with a much broader scope, as it intends to address the same type of problem but for the new scenario of the "Fourth Industrial Revolution". Its goal is to revolutionize the manufacturing and production industry by integrating the Internet of Things (IoT), Cloud Computing, data integration and other technological advances into the heart of production and manufacturing systems. This "Industrial Internet of Things" concept utilizes both Cloud Computing and the Internet of the "things" to take processes that were normally managed internally by both people and machines, and move them into the "cloud" where they can be managed from anywhere in the world.

Within a manufacturing system, those "things"—the so called Cyber-Physical Production Systems (CPPS)—are becoming more "complex" as they are not just machines "thinking in the cloud", but also machines producing "mountains" of new data. This poses new challenges: what to do with those "piles of data" being collected (big-data) and how to handle them, i.e., how to accurately process and "extract" useful information from it, in almost real-time ?

In the generalization of this study for an Industrial Manufacturing complex scenario, in which several factories and depots are considered, the processing of the "big-data" can be tackled with a parallel and distributed execution method, using cloud-computing technologies. Indeed, this type of environment gives not just advantages of cost, and fault tolerance, but also makes it relatively easy to transform existing inherently computing-intensive statistical sampling problems to a larger scale, and processed in real-time (or as little time as possible).

With the mathematical problem completely solved, our objective is now to embed it in an appropriate computational environment, with a Proof-of-Concept (PoC) solution deployed in the public Google Cloud Platform (GCP), for the accurate estimation of the statistics.

This paper is organized as follows. Section II presents the generalized problem of estimation of μ and σ for very large datasets. Section III describes the parallel processing strategy to be used. Section IV describes the PoC cloud-based implementation of the statistics tool, with a basic evaluation in Section V, and Section VI concludes with a summary and future works.

II. Estimation from sums

In the fertilizer industry, the granular products are sold in bags, of e.g. 25 kg, and the customers bring their own trucks to transport the load of bags, the number of bags in which meeting the customer's necessity. Thus, the number of bags in each truck will be typically different. (The particular case of equal numbers of bags is trivial.) The trucks are weighed upon arrival ("empty") and departure ("full"), so the customer's load will be the difference. To make the situation clear, suppose, for example, that a customer wishes to buy $n = 200$ bags of a certain fertilizer, sold in bags of weight $m = 25$kg. This load should correspond to $n \times m = 5\,000$ kg, but, due to intrinsic variability, the real weight would be expected just to be near that value, not the exact one (rigorously, a factory targeting its bag weight to 25 kg, if this were the nominal weight, would be in trouble, for its fraction defective would be near 50% !). Therefore, the value obtained in practice is verified and recorded as the last step in the business control. (The subject of the tolerances in this verification is essential, but not addressed here.) This value is kept for manufacturing and legal reasons. We assume, as is usually done in reality, that the random weight of each bag is Gaussian, with, say, a certain $\mu = 25$ kg and $\sigma = 0.2$ kg, values that are, otherwise, realistic in the fertilizer trade. Now, apart from the necessary business control, how can the observed truck weight data be used to verify the correctness of the upstream manufacturing process? We mean, how can these final data help estimate the parameters μ and σ of the distribution of the weight of the individual bag?

The historical setting that we observed in that factory is now generalized. Indeed, the factory had not one delivery channel but 20 distribution depots, one near the capital city of each region of the country—18 "distritos", in continental Portugal, plus 2 in the Atlantic archipelagos, Azores and Madeira (a "*Distrito*" i.e., a district, province, is the basic administrative region in Portugal, averaging about 5 000 km^2). Thus, the final data came from 20 sources. Instead of one list of final weights, there were 20 lists, each resembling the fictitious numbers given in Table I.

TABLE I: TRUCK LOADS IN A CERTAIN DEPOT, WITH (CALCULATED) AVERAGES.

Truck, t	**Number of bags, n_t**	**Weight (kg), m_t**	**Aver. (kg), $\bar{x}_t = m_t / n_t$**
1	5	124	**24.8**
2	10	253	**25.3**
3	20	504	**25.2**
4	30	747	**24.9**

The table might show the 4 sales in, say, one week: a customer bought 5 bags, weighing 124 kg; another customer bought 10 bags, weighing 253 kg; etc.. In the table and henceforth, we use the notation: $t = 1..T$, with T the number of customers (trucks); n_t is the size (number of bags) of each load; m_t is the weight of each load; and $\bar{x}_t$ is the calculated (bag) weight average, in a 4.th column added (not existing and unnecessary in the real lists). (Our notation $m..n$ means all the integers in the range of the integers m–n, both included, as inspired in certain computer languages). The average was found to be the statistically suitable key measurement in our computing.

The estimation of the mean (μ) is obvious, not so for the standard deviation (σ). For the mean, in this example, the estimate is simply:

$$(124 + 253 + 504 + 742) / (5 + 10 + 20 + 30) = 25.05 \text{ kg}$$

(A confidence interval would be in order, not only for the mean but also, below, for the standard deviation, but that is not in the objectives of this paper.)

Let us now, for convenience, define the “weight”, w_t, of each load, with $N = \sum n_t$, leading to Equation {1}.

$$w_t = n_t / N \qquad \{1\}$$

Then, the estimates for μ and σ (not proved here) become the following weighted statistics, respectively, in Equations {2} and {3}.

$$\hat{\mu} = \sum_{t=1}^{T} w_t \bar{x}_t \qquad \{2\}$$

$$\hat{\sigma}^2 = \frac{N}{T-1} \sum_{t=1}^{T} w_t \left(\bar{x}_t - \hat{\mu}\right)^2 \qquad \{3\}$$

Let us, now, generalize the question. Instead of one depot, there will be many (above, 20 were mentioned). We will keep the same business period, e.g., one week. Each depot will generate its list of records. Each list will have its data, in a file that is here considered large, containing:

N, number of records

T, cardinality of different customers

n, load size vector

a, many (*N*) average weight vectors

For simplicity in this text, *N*, *T*, and ***n*** will be kept equal for all the depots, although of course all these values will obviously differ for every depot, according to the business intensity.

The depot files will be sent from each depot to a central receiving system, which will then compute the desired estimates of the mean and standard deviation of the production of bags, in our example.

The fundamental statistics for the computing is, thus, to take advantage of the list of records to estimate the essential parameters of the (Gaussian) distribution of the individual bag weight.

III. Computing Strategy

The main “problem” at hand is that we cannot use a single computer to process large amounts of data (as it takes too long to process), and the solution is to make use of *Distributed computing*, i.e., the simultaneous use of many interconnected computing nodes (processor, and memory independent) to perform very large-scale computations and process very large datasets, desirably capable of integrating new data and updating existing results without having to re-compute everything. For this scenario, the first step is to break a task into sub-tasks and distribute them to different nodes. At the end of the process, the output of each node is aggregated in order to have final output.

One of the more intuitive approaches for the case of calculating the statistics on large datasets, such as the mean, μ, and the variance, σ^2 of a set of numbers, is the MapReduce framework [5], a methodology for standardizing the method of implementing massively parallel data processing. A MapReduce-based implementation would slice up the data and process the slices in parallel, with the bulk of the actual parallel processing occurring in the “map” step. The “reduce” step is usually a minimal step to combine the independently calculated results, i.e., to implement a way to combine the statistics in parallel (for example, the means of two samples, and find their combined mean).

A. Distributed Computing Frameworks

The Apache Hadoop (https://hadoop.apache.org) framework consists of an implementation of the MapReduce method combined with the Hadoop Distributed File System (HDFS), as illustrated in Figure 3. The input data, that typically consists of key-value pairs is stored on the HDFS. They are split into fixed-size blocks, and allocated to the user-specified “maps”. The “map” function is then applied on the input block to produce intermediate key-value pairs. The intermediate data is partitioned by the key, and the grouped records are shuffled to the appropriate “reduces”. Then, the shuffled records are merged and sorted in the node that a “reduce” task located. Each “reduce” sequentially processes key-value pairs by the user-specified “reduce” function to generate the final output key-value pairs.

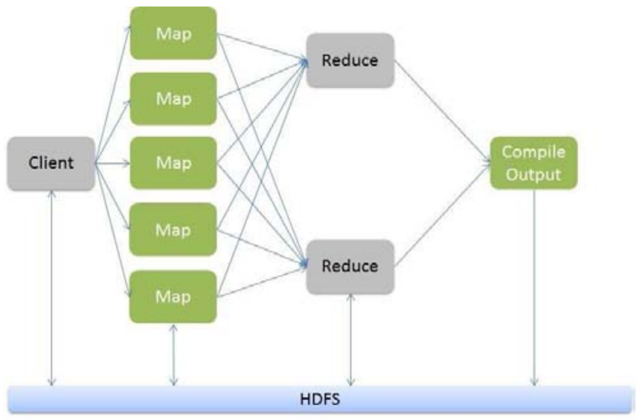


Figure 3. The Apache Map-Reduce model

Although, Apache Hadoop is widely used for fast distributed computing, it has the disadvantage of not using “in-memory computation”, which is keeping the data in RAM instead of Hard Disk for fast processing (which is the enabler feature for big-data). In order to understand the advantage of in-memory computation, suppose there are several “map-reduce” tasks happening one after another. At the start of the computations Hadoop reads the data from disk for mapping, then performs the “map” operation and saves the results back to disk. In the next step, the “reduce” operation, Hadoop reads the saved data from the disk in order to produce the output which is again saved to disk for further steps. These read-write operations introduce huge delays in the computation.

When the Apache Spark (https://spark.apache.org) analytics engine framework was developed, it overcame this problem by using In-memory computation. The Apache Spark is a fast cluster computing framework based on in-memory computation, which is a big advantage over Hadoop, as it can run tasks up to 100 times faster, when it utilizes the in-memory computations, and 10 times faster when it uses disk, than traditional map-reduce tasks. Figure 4 shows the comparison of Hadoop MapReduce versus Spark, for the same type of tasks. Apache

Spark, however, is not a "replacement" of Hadoop as it is actually designed to run on top of Hadoop. In fact, the Spark framework extends the MapReduce model to support more types of computations using a functional programming paradigm, and it can cover a wide range of workflows that previously were implemented as specialized systems built on top of Hadoop.

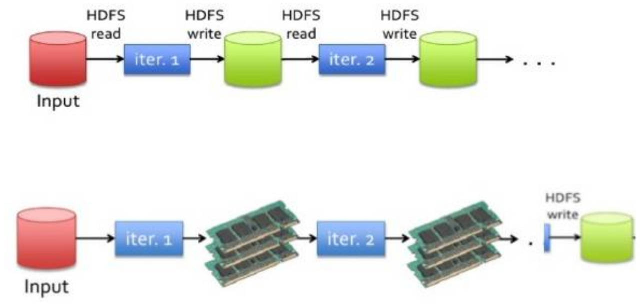


Figure 4. Hadoop MapReduce (top) vs. Spark (bottom)

The key idea of distributed computing with Apache Spark, as previously stated, is that a computation is split into pieces that can be run independently on multiple computing nodes. In Spark, the "Driver Program", illustrated in Figure 5, initiates the computation.

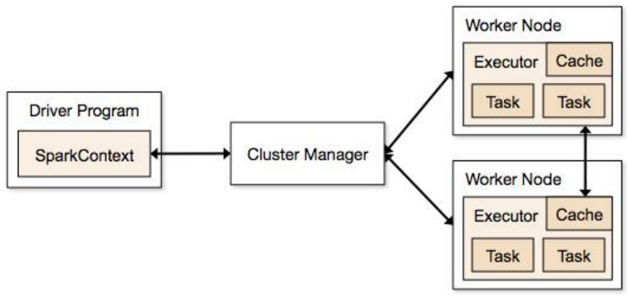


Figure 5. Apache Spark architecture.
Image source: https://spark.apache.org/docs/

It receives permission form the Cluster Manager to launch the "executor process" on worker nodes. Each "executor process" completes one part of the computation. The result of each "executor process" is then communicated back to the "Driver Program", which integrates the results and makes them available to the user.

The key idea in Apache Spark is the Resilient Distributed Dataset (RDD), illustrated in Figure 6. This is a dataset that is stored redundantly in memory across multiple computing nodes, so that if any node goes down, the RDD can be recovered. The RDD's are *immutable* (*read-only*), meaning that they cannot be changed after being created and are typically the result from either a load from disk, for example, by starting a file in HDFS, or the parallelization of an existing collection, followed by application of transformations. To make use of an RDD, the program applies an *action* to it. Most *actions* return a result to the "Driver Program". For example, in a MapReduce method, a "map" (a transformation) performs an operation on each element of an RDD and returns a new RDD. However, in case of a "reduce" (an action), it reduces/aggregates the output of a "map" by applying some functions, i.e., computations (reduce by key).

An *action* can therefore be something such as summing the numbers in the RDD, or counting the elements that meet a certain condition, and multiple actions can be combined for complex tasks. Each RDD is an ordered collection of records, which are split into *partitions* that may be stored on different computing nodes, and since an RDD is read-only, there is no need to synchronize changes to the RDD within or across computing nodes. The Directed Acyclic Graph (DAG) Scheduler, illustrated in Figure 6, reflects the dependency relationship between RDDs.

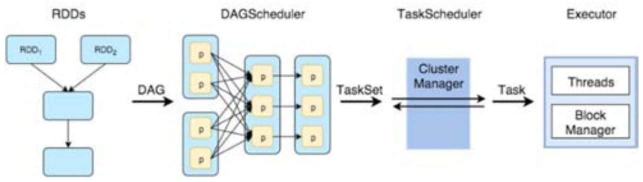


Figure 6. Spark Scheduling Process

Spark may also share values of variables, constants and functions to be used in parallel operations. Two types of shared variables are supported: *broadcast variables* that can cache values in memory on all nodes, and *accumulators* that are temporary variables such as counters and sum registers.

With this model of Distributed Computing provided by the Apache Spark framework, it will be possible to implement the processes of our scenario for the accurate estimation of the statistics.

B. *Accurately Computing Statistics on Partitioned Samples*

Conventional algorithms for statistics estimation were not designed around memory independence (parallel processing) or "shitted data", i.e., data collected dynamically, as they are typically "two-pass" algorithms, i.e., passing through the whole data twice, as is the case of computing the mean (1.st pass) and the variance (2.nd pass) [6].

The so called "online", or "one-pass", algorithms try to avoid the "two-pass" nature of standard algorithms, and are based on the idea of keeping the data available to serve requests, rather than having to perform "offline" computations, being thus capable of integrating new data and updating results without the need to re-compute everything.

There are, however, issues with some formulations used for computing the variance involving sums of squares, namely in MapReduce methods, that can lead to numerically instable or inaccurate results, as well as to arithmetic overflow when dealing with large values, suffering therefore from precision loss and even resulting in negative variances, meaning that *catastrophic* cancellations have occurred.

To overcome this type of issues and potential inaccuracies, we implemented a "parallel algorithm", developed by Welford [7] and updated by Chan *et al*. [8], that is capable of merging multiple sets of statistics calculated online, which works for any partition of the dataset.

The approach is to exploit the fact that we can operate efficiently within a partition of an RDD in Spark, and so, it becomes easy to calculate the mean and variance within each partition. Since the partitions may contain unequal numbers of data values, these will need to be weighted means and variances.

IV. Implementation

As previously mentioned, the PoC of the computational environment was deployed in GCP (https://cloud.google.com), by means of an "Infrastructure as Code" method, using Terraform as Automation engine (https://www.terraform.io), a tool for building, changing, and versioning infrastructures (networks, services, compute instances, storage), together with Ansible (https://www.ansible.com), a Configuration Management tool for configuring the compute instances of the infrastructure.

Terraform codifies Application Program Interfaces (APIs) into "declarative" configuration files for the target environment, while Ansible performs its actions, between a "procedural" and "declarative" language through *ad-hoc* commands that allow for procedural-style configurations in the compute instances.

The architecture of the computational environment, considers a Front-end system and several intermediate nodes responsible for the Spark-Hadoop parallel distributed computation environment, as illustrated in Figure 7.

The Front-end node includes a "welcoming" Web page containing some description of the *job* to be triggered, and with configurable parameters for functionalities and data input, and a "results" web page, for displaying the statistical results of the computations, such as mean, variance, standard deviation, etc..

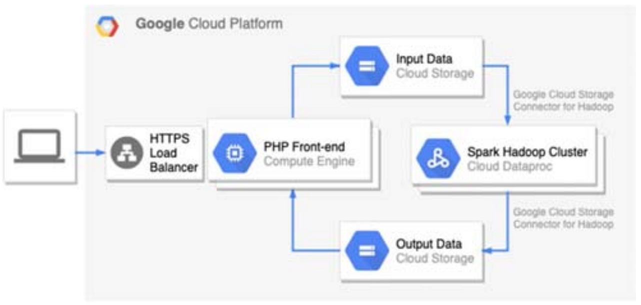


Figure 7. The GCP computational Environment

The intermediate nodes are an abstraction of a cluster infrastructure, in terms of the end-user, as they are called for the computation *jobs* through some API calls or framework calls (in our case, these are PySpark API calls written in the Python language).

Each *job* triggers a Spark Job that uses *worker nodes* of the Google Cloud *Dataproc engine* (in our case, the Spark-Hadoop cluster, e.g., a cluster with one master and three workers) to perform the parallel computations.

The PHP WebApp developed for the PoC consists just of a PHP Standard Environment for the very simple Front-end web pages, which also allows for making the API calls to Google Cloud *Dataproc* engine. The Front-end subsystem runs in three private Nginx (https://www.nginx.com) virtual servers deployed in the Google *Compute* Engine and with a HaProxy (http://www.haproxy.org) Load Balancer, deployed in another virtual server.

When a *job* is finished, the results are stored in a Google Cloud Storage Bucket that is then accessed by the Front-end to display all the information to the end-user.

A simple "procedure" of generating a job is as follows:

1. First, the application creates a *SparkContext* instance.
2. Then, the application generates the RDD using the *SparkContext* instance.
3. Through a series of transformation operations, the original RDD is transformed into a RDD of another type.
4. When an action operation acts on a transformed RDD, it will call the *runJob* method of the *SparkContext* instance.
5. The *runJob* call is the starting point of a chain of following actions.

At this stage, the system is just a PoC, and so, the solution still requires some tuning and optimizations, both at the infrastructure level and at the "code" level, in order to allow more flexibility in the specification and type of jobs, as well as for allowing to compute other statistics from the same data, or producing graphical outputs.

V. Evaluation

Some preliminary and very basic tests were conducted with the PoC system to demonstrate that it works as desired, considering that the solution is just an initial approach, not optimized. The obtained results were nevertheless accurate, when compared with a computation with the same datasets performed in a desktop computer with a 2.5 GHz Intel Core i7 CPU and 16 GB of RAM, running a *nix operating system.

The obtained results were the following:

• Calculation of mean and variance from a small dataset containing just 10 entries, to check if the results were right.

• Calculation of mean and variance from a big dataset containing 1 Million (10^6) entries: a runtime of 30 seconds.

• Calculation of mean and variance from a huge dataset with ~90 Million (9×10^7) entries: results were also accurate, but with a runtime of ~540 seconds (~9 minutes).

VI. Conclusions

This study, based on a concrete problem in a fertilizer factory, had as an objective to estimate statistical parameters of a process, dealing with weights of loads of unequal (known) numbers of bags, with many distribution depots, the data from which have to be collected for processing. The problem is addressed in a Cloud Computing, big-data framework, using Apache Spark, with its "in-memory computation" and Resilient Distributed Dataset. The computation uses Terraform and Ansible, and was deployed in Google's Cloud Platform. It is triggered on a Web page in PHP, leading to results of the expected statistical calculations. The evaluation preliminary tests confirmed accuracy and produced low runtimes.

ACKNOWLEDGMENT

R. Cruz thanks the Department of Computer Science and Engineering, IST (“Instituto Superior Técnico”), “Universidade de Lisboa” (University of Lisbon, Lisbon, Portugal), and INESC-ID, with support from ”Fundação para a Ciência e Tecnologia” (Portuguese *National Science Foundation*), through Project UID/CEC/50021/2019.

M. Casquilho thanks the Department of Chemical Engineering, IST, and CERENA, “Centro de Recursos Naturais e Ambiente” (Centre for Natural Resources and the Environment), both of IST, with support from ”Fundação para a Ciência e Tecnologia” (Portuguese *National Science Foundation*), through Project UID/ECI/04028/2019.

The basic computing was done at DSI (Direcção de Serviços de Informática) the computer and network services at IST.

REFERENCES

[1] M. T. Barros, and M. Casquilho, “Sigma (σ) from sums of unequal size samples, a conjecture: heterogeneous servers, over the Internet”, CISTI’2018, 13.ª Conf. Ibérica de Sistemas y Tecnologías de Información (13.th Iberian Conference on Information Systems and Technologies), Cáceres (Spain), 2018, 13–16 Jun..

[2] M. Casquilho, “Sigma (σ) from sums of unequal size samples”, http://web.tecnico.ulisboa.pt/~mcasquilho/compute/qc/Fx-3sigmaFromSums.php , accessed 01-Feb-2019.

[3] V. M. Ponce, http://ponce.sdsu.edu/ , accessed 01-Feb-2019.

[4] V. M. Ponce, “Vlab”, http://ponce.sdsu.edu/online_calc.php , accessed 01-Feb-2019.

[5] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” in OSDI’04: Sixth Symposium on Operating System Design and Implementation, (San Francisco, CA), pp. 137–150, 2004.

[6] T. F. Chan, G. H. Golub, and R. J. LeVeque, “Algorithms for computing the sample variance: Analysis and recommendations,” The American Statistician, vol. 37, no. 3, pp. 242–247, 1983.

[7] B. P. Welford, “Note on a method for calculating corrected sums of squares and products,” Technometrics, vol. 4, no. 3, pp. 419–420, 1962.

[8] T. F. Chan, G. H. Golub, and R. J. LeVeque, “Updating formulae and a pairwise algorithm for computing sample variances.,” in COMPSTAT 1982 5th Symposium held at Toulouse 1982, (Physica, Heidelberg), pp. 30–41, 1982.

CERENA

FCT Fundação para a Ciência e a Tecnologia