BigDataNetSim: A Simulator for Data and Process Placement in Large Big Data Platforms

Leandro Batista de Almeida*, Eduardo Cunha de Almeida†, John Murphy*, Robson E. de Grande‡, Anthony Ventresque*
*Lero, School of Computer Science, University College Dublin, Ireland
†UFPR, Federal University of Parana, Brazil
‡Department of Computer Science, Brock University, Canada

***Abstract*—Big Data platforms are complex distributed systems running many processes/threads on multiple machines. Writing programs (jobs), setting-up clusters, tuning the many parameters of these Big Data platforms is a skill- and labour-intensive task and many tools have been proposed to help developers and engineers. However there is nothing that combines both a good modeling of the behaviour of Big Data platforms at scale (including data and job placement, networking elements) and an accurate representation of Big Data jobs. In this paper, we propose BigDataNetSim, a simulator of Big Data platforms that models accurately the main components of the Hadoop ecosystem (e.g., HDFS, YARN) when deployed in a large scale system. BigDataNetSim can help Big Data engineers set-up their systems, prototype jobs and improve components/algorithms of the core Big Data platforms. In particular, we show that BigDataNetSim simulates a real Hadoop cluster with a high degree of accuracy in terms of data placement and job placement, and that BigDataNetSim can scale to very large systems.**

***Index Terms*—Big data, Hadoop, simulation.**

I. Introduction

A. Background

Simulating Big Data platforms (e.g., Apache Hadoop) is an important research and engineering challenge that has been the focus of a lot of attention for the past decade. These Big Data platforms contain lot of parameters that have to be tailored to particular infrastructures, data sets and algorithms and big data jobs are somehow difficult (read: "not intuitive") to engineer. This problem seems well suited for using simulators, that can help developers of Big Data platforms as well as users (Big Data engineers and Data scientists alike) to tune the various parameters, chose the correct algorithms and do the proper resource allocation.

Many of the simulators we find in the literature focus on prototyping Big Data jobs though, e.g., MRSim [1], SimMR [2], MRSG [3], Mumak [4]. This is interesting in itself, as designing and implementing Big Data jobs is not an easy task, but this does not address the critical "distributed" nature of these systems.

Moreover, when they consider the deployment of jobs, the Big Data platforms' simulators often focus on one single element of the large scale distributed systems that Big Data platforms are: (i) the job placement/scheduling (e.g., BS-YARN [5], SimMapReduce [6]), YARN Scheduler Load Simulator [7]); (ii) the VM provisioning (e.g., MR-CloudSim [8], HSim [9]); or (iii) focus on optimisation of deployed systems (through a model based and/or performance oriented approach: e.g., Starfish [10], Doopnet [11], DAGSim [12] and Camp [13]).

B. Research Challenge

'Big Data' is one of the most important IT concepts of the past decade, in the industry and in research – as the amount of data generated and processed has become very large and the classical solutions become impractical [14], [15]. Big Data platforms (e.g., Apache Hadoop) are complex distributed systems designed carefully to process big data sets – i.e., data sets that are extremely large or extremely fast. Distributed systems are inherently complex as they are an intricate collection of interconnected machines that interact at various levels of their hardware, networking and software stacks. As such, setting up a Big Data platform can often be seen as a work of art, with capital allocators, data engineers and data scientists working hand-in-hand to tailor their systems to their needs. For instance, large multinational companies often publish papers [16], [17] and blog posts [] describing the scale and complexity of their systems, as if they were proud of their 'masterpieces' – or if they wanted to use it to demonstrate that their systems are good to potential candidates (According to a recent survey published by the multinational HR consulting firm Randstad [18], Big Data engineer was the "best in-demand job for 2017"). On top of that, the algorithms to process Big Data sets are also skill- and labour-intensive [19]: to address this challenge, large companies build special data centres/clusters to run their complex algorithms (often on non linear data structures), complex data sets are analysed using multiple variants of sophisticated algorithms, etc.

In short: managing data at this scale/speed is challenging in terms of infrastructure (i.e., "where to store the data and how to access it?") and processing (i.e., "how to efficiently run parallel and distributed jobs?"): there exists a plethora of sophisticated Big Data platforms, each designed to address a type of data structure, data set or algorithm, with multiple parameters.

Hence the research challenge we address in this paper is the possibility to build a simulator that: (i) has the ability to model real Big Data platforms with a high degree of accuracy,

(ii) capture the data movement/migration (i.e., when data is transferred between nodes in the Big Data platform) with a high degree of accuracy and (iii) scales up to large clusters of Big Data platforms.

C. Example

As a motivating example, consider a group of Big Data engineers or architects who need to set up a cluster of machines for some graph processing using Apache Giraph [20] (i.e., Big Data platform for non linear data structures based on the Hadoop ecosystem and the Bulk Synchronous Parallel concept introduced first by Pregel [21]). They will have to select the best infrastructure for their needs (i.e., How many racks? How many machines? What type of network interfaces?) as well as the data distribution (e.g., what partitioning algorithm?) and the job/task distribution when running their algorithms (e.g., which nodes to use for a particular algorithm/process?).

A lot of these questions get default answers by the different elements of the Big Data stack: e.g., there is a default partitioning algorithm in Giraph, there is a default replication factor in HDFS, there is a default job/task scheduling algorithm in Hadoop. But there is a lot of parameters and options in these systems and performance can be impacted by a lot of elements. Tools, such as, Starfish [10], try to model the performance of the different elements and to adapt the parameters of the Big Data platforms (from the infrastructure to the various scheduling and placement algorithms) and to help the users of such systems.

However, what developers of Big Data platforms, and advanced users alike, would like to have is a simulator that tells them what to expect with the various algorithms implemented in the Big Data platforms, especially *at scale*. For instance, these developers would likely use a simulator that shows them what is the best data placement or task placement algorithm for a certain configuration, or which network topology can provide more performance.

D. BigDataNetSim

In this paper we propose BigDataNetSim, a fully functional[1] simulator of data and process placements, as well as networking/data transfer, for large scale Big Data platforms. BigDataNetSim is able to simulate most of the components of the Hadoop ecosystem (we have implementation already done for HDFS, YARN, MapReduce; and it is easy to add new components). BigDataNetSim has very good performance and scales to large (realistic) clusters and can be configured in many ways (different network topologies, placement algorithms etc.).

Our contributions in this paper are:

- We propose a novel simulator for Big Data platforms, capable of running Big Data jobs on Big Data platforms and to analyse the performance (e.g., execution time/throughput) of the modeled platforms and jobs.
- We perform a thorough evaluation of BigDataNetSim using a real cluster of 20 machines in 2 racks. We compare the data placement of HDFS (the data management component of the Hadoop stack) and the job scheduling/placement of YARN (the process management component), and we show that BigDataNetSim is a very accurate model for real systems (between 2.9% and 9.6% difference on average for data placement, depending on the file size, and 4.4% difference on average for job placement). We also evaluate the scalability of BigDataNetSim and show that we can simulate easily various classical Hadoop jobs.

[1]Code will be provided on Github upon acceptance of this paper.

The rest of this paper is organised as follows: Section II describes the simulator and its architecture, Section III describes the set-up for the evaluation tests, Section IV discusses the evaluation results and Section V concludes this paper.

II. BigDataNetSim Architecture

BigDataNetSim is a simulator designed to execute analysis of data placement, task placement and network parameters of a Big Data cluster based on Hadoop (HDFS/YARN) and data processing engines running on top of it, like MapReduce, Hive or Spark. BigDataNetSim is intended to simulate large cluster configurations, with several thousands nodes, allowing the analysis of network and data processing engine parameters on those large clusters.

The simulator development was based on Java language, making available command line and web based interfaces. Simulation parameters describe the cluster, the environment and the network connecting the nodes. The following sections will describe the requirements for the simulator and its architecture, internal operation and usage.

A. Requirements for BigDataNetSim

As BigDataNetSim was developed to support certain research tasks, regarding data and task placement, as well as network topologies and protocols, its requirements are based on the type of analysis that research need to perform, which scenarios and tests will be executed, and which data is needed as a result from the simulations.

1) Which data we need from the simulator?: In general, the simulator requires the following:

- Distributed File System: information about the files stored in the file system, the location of the file blocks among the nodes, number of replicas.
- Executing Tasks: how many tasks each job generates, where are the tasks running, from where each task is reading the data, classification of a task based on the location of data to read (local, rack, external).
- Network Infrastructure: how much data is passing through the network, how the network topology and the switching protocol work together, how much time a particular amount of data will use to be transferred in the network segments.

We answer these questions in detail in the next sections.

2) Scenarios to simulate: The typical scenarios we have in mind reflect real clusters where 1000s of machines is not unusual [22]. In terms of network organization, Hadoop configuration only registers in which rack each node is. In addition to that, in the simulations, we need to select which topology will be used on the cluster (hierarchical, FatTree based, new proposed topologies), and which switching protocol will be used among the switches in the network (Spanning Tree, Shortest Path Bridging, SDN based protocols). These features are complex and expensive to implement in a real cluster, so the simulator is important to help in the selection of the most appropriate and viable set of features, in a simulated environment, and later progressing for the physical tests with the best solutions.

As mentioned before, the reading part of a big data job is the only constant among the different tools and frameworks, so the research is focused on this particular phase as it could be costly to simulate all these frameworks. Still, as the reading part is a costly one in terms of resources and time, optimizing just this part can result in significant gains for the job processing, becoming a generic way of improve any available tool on top of HDFS. In this reading phase, the focus is on the data movement, affected for the data and task placement, and the underlying network structure, including the topology, protocols and network features. And as a result of this focus on the reading phase, this simulator is not required to model the remaining phases of a big data job.

3) Remote Reading in Big Data Platforms: As the data is distributed in a big data cluster, so is the task of reading those data, considering that HDFS is a common base for several big data tools and frameworks [22].

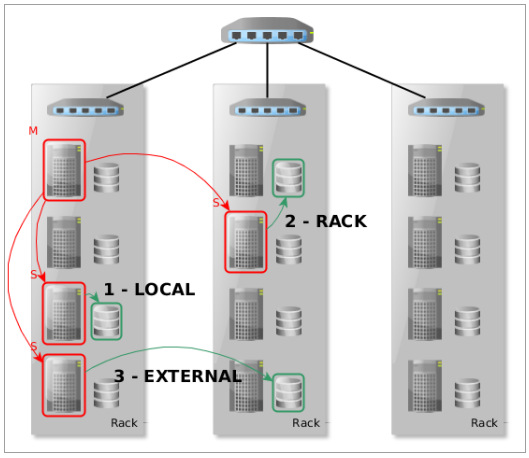
Several cluster processing engines can be used over HDFS, but YARN is a very common one, and supports several frameworks, like MapReduce, Hive and Spark. Besides, YARN is the default processing engine available in Hadoop, making easy for other tools to use it.

In general, big data platforms are based on the ”move the code, not the data” concept, meaning that when possible, it is desirable to process data in a local way, transferring a piece of code (very small, compared to data) to a node, and then reading and processing data blocks stored on that particular node. As each node has limited resources to start containers to execute the code [23] (usually related to the number of available cores on the node), there is also a limit of how many simultaneous tasks can be executed at a given time in a specific node.

In real cluster when the number of jobs and tasks is large, it is likely that processes cannot always be executed on the machines that are storing the data blocks. Hadoop does not have a very sophisticated algorithm for task placement, as decisions have to be taken quickly and optimising task placement is a difficult (NP Hard) problem - which makes the issue even more prevalent. Eventually there will be a divide between data blocks and tasks that creates a large overhead on the Big Data platforms' performance.

This is the concept of “data locality” [23], that Hadoop tries to keep when possible. Following this concept, the tasks can be classified as “local”, when reading data from the local storage, “rack”, when accessing data from a different node in the same rack, and “external” when reading data from another node in another rack, as shown in Figure 1. As expected, the network latency increases when data locality decreases.

Fig. 1. Local, same rack and external reads.



In a cluster with a low level of usage, and no concurrent jobs using the same input data, it is expected that the majority of the tasks could be local, affected only by the processing engine strategy for task placement. However, the usual conditions for a cluster points to a scenario when several applications and jobs are using the cluster at a given point in time, and several jobs are probably using the same input information for its processing. In this case, the processing slots or containers could be exhausted for part of the nodes, forcing the processing engine to find an available container in a node other than one containing the input data. This process is increasing with the number of concurrent applications and jobs running in the cluster. In the limit scenario, with all the containers in all the nodes being used, it is expected that the majority of tasks run as external ones. This poses a heavy load in the network infrastructure, hence the importance in properly measure that.

B. How Does the Simulator Work

In general, simulations are used in the initial phase of a research or test set, to select ideas and concepts, testing it in a quick and inexpensive way.

So, in these cases, the common use case for our simulator in our project is to generate a cluster of a particular size, with a particular network topology, create the file system in it, populate the file system with files of a specific size, submit a given number of concurrent jobs, generate the metrics for the network usage and generate the reports about the simulation.

The details of how the simulator accomplish that are described in the following sections.

1) Parameters for the Simulations: For the simulations necessary to the ongoing research, we selected a set of parameters that best represented the real scenario, with the appropriate

level of simplification, in the sense that not all the parameters available for a Hadoop cluster make a significant difference to the results we need.

In a general way, the parameters can be divided into three main categories, the cluster structure (including distributed file system), network structure and frameworks structure, as described below. In addition to the parameters that describe the scenario, there is also a set of parameters that describe a test execution, and those will be described at the end of the section.

Cluster configuration: Number of nodes in the cluster, number of nodes per rack in the cluster, number of available processing slots in each node, nodes with different configuration in the number of processing nodes. The simulated file system can configure the size of a file block, the level of replication of each block and the read rate for the storage, including a value for overhead. All these parameters are discrete values, and can be set for each execution. For HDFS, the block placement policy can be configured, either using the default one, or a custom one, implemented using the simulator structure.

Network configuration: Network link bandwidth (classified by the topology layers), link delay, switch delay, network overhead, frame size, MSS (maximum segment size). Based on these parameters, the effective link bandwidth can be calculated. Apart from these discrete values, the simulator can configure other behaviors in the network: switching protocol (STP, SPB, SDN oriented, custom) and the topology. The topology can be set as a regular hierarchical topology, like a FatTree, or as a custom topology, defined using a graph. In either way, it is possible to configure the number of switches in each level or place and the connections between the switches, including redundant and loop connections.

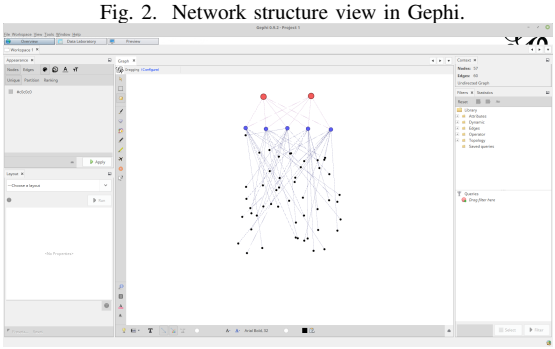
Frameworks and policies/algorithms: These parameters control how the jobs will execute on the simulator, and have a direct impact on how the tasks will be distributed among the nodes, hence their importance. Job processing engine: the simulator have one processing engine modeled: MapReduce over YARN. Others are in project to development, like Spark over YARN and Spark over HDFS. For YARN, the capacity scheduler is modeled, others will be modeled in the future. For MapReduce engine, some details are modeled, like turning on and off the speculative execution .

Test execution parameters: Number of concurrent jobs/users, including the range of concurrent jobs (from 1 to 256, for instance) and the step used (from 8 to 8, for instance). Number of tries/rounds for each test, to diminish any deviation caused for the several random aspects of the tests, this parameter is usually configured for at least a few dozens.

2) Interfaces for the simulator: The amount of available parameters and the number of generated reports can make the simulator operation complex, so user interfaces were developed to help each major use case to fulfill the objectives with the minimum overhead in operation.

The simulator was designed to be used mainly in two ways, as an exploratory tool, executing quick tests in different scenarios with different configurations, and as a batch tool, to execute a massive amount of tests for a particular configuration. Those two kinds of usage leaded to two main ways of using the simulator, through a Web Interface, and a CLI (Command Line Interface).

Fig. 2. Network structure view in Gephi.



About the exploratory tests, these are used to quickly test a given configuration, to check if that particular setup is reasonable or not for a longer simulation. Those tests are often executed with reduced parameters and point to more extensive tests, once the proper configuration is decided. For this kind of test, the web interface was develop, with a predefined set of parameters and a intuitive interface to show quick results.

Follows a brief description of each interface and auxiliary tools:

Command line interface: The main interface for using the simulator is the command line, and the parameters can be set either as a JSON configuration file, or a list in the command line itself. This allows to the simulator to be included in larger batches of scenarios, to be executed as a whole, in a script. The results will be shown in the standard output, and saved to CSV files, if requested.

GUI helper interfaces for network design: There is a particular complex parameter in the simulation process, the network topology design. Besides the conventional hierarchical topology, the simulator allows any custom designed topology to be used as the base infrastructure for the cluster network. For this reason, the simulator can export a graph file in a GEXF format [24] to allow the analysis of a particularly complex topology in graph analysis tool like Gephi, as shown in Figure 2. For the same kind of complex topology, was developed a GUI interface that shows the resulting graph, allowing minor changes in the topology, like adding more connections between the Top of the Racks switches and the core switches, and increasing the density of links between the core switches, as shown in Figure 3. This GUI tool needs to be customized for a particular kind of network topology.

Web interface: As mentioned, the web interface was developed to allows quick exploratory tests, with a subset of parameters, in order to find proper configurations for more extensive tests. It was alse developed to allow the simulator to be used by a larger team, and taking advantage of more processing power available in a proper server, rather than the user own computer. In this case, the results will be displayed

Fig. 3. Network structure design.

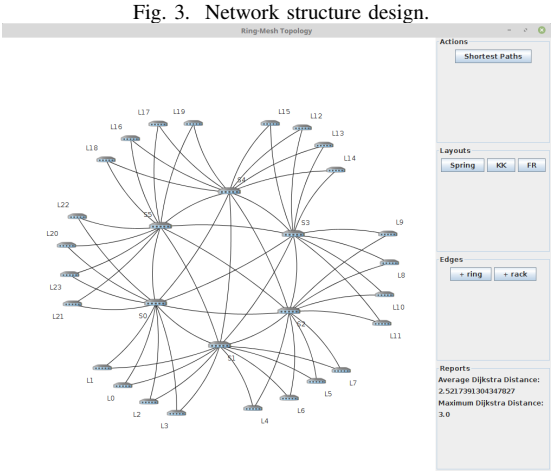
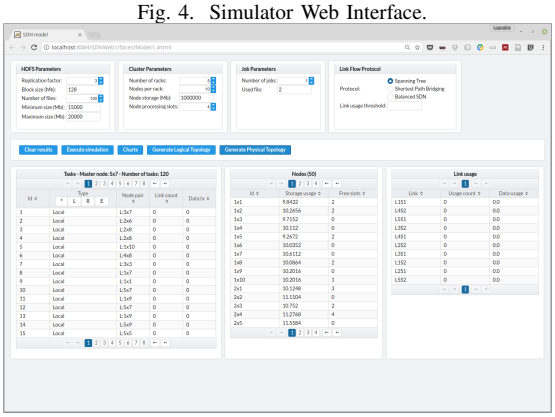


Fig. 4. Simulator Web Interface.



as web reports, in addition to the CSV files. The web interface is shown in Figure 4.

Configuration files: As the number of parameters grow, it can be prone to errors and mistakes when using command line parameters. For this, the simulator can use a configuration file for the input parameters, in the command line interface. The configuration file can be expressed in two formats: a Java Properties file or a JSON file, with the named parameters and the corresponding value.

Results output: The simulator interface can return the information in text reports in CSV format, and charts using the popular GNUPlot format. Execution parameters can configure which of the reports will be generated in each simulation run, and the detail level needed.

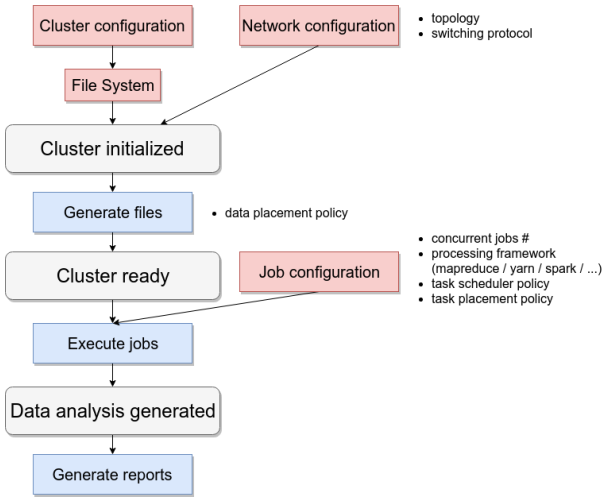
C. Phases of a Simulation

In this To better understand how a simulation is conducted in the simulator, it is interesting to detail the steps of a simulation task, and how these steps are feed with the parameters and relate with each other.

In a general way, a simulation follows a workflow with this main phases, as shown in Figure 5:

- Cluster configuration: when the parameters for the file system, network topology and cluster are set, preparing the environment for the execution. Data placement strategy is also configured in this phase.
- File system generation: generation of the files, according with size and number parameters, and using information from the cluster and network configuration to apply the data placement strategy.
- Job configuration: parameters for the job execution are set on this phase, including the number of concurrent jobs, which file (s) will be used as input, which processing framework and its parameters and the strategy for task placement and scheduling.
- Simulation execution: execution of the simulation according with the configured parameters while storing the intermediate results.
- Report generation: aggregation of results and generation of the requested reports.

Fig. 5. Simulation execution workflow.



In the workflow, a critical phase is the process to calculate the amount of network traffic for the jobs execution, as described in the Algorithm 1. This values are particularly important for the research that started the simulator development, as network optimizations are the goal for that. In this sense, the information needed is the amount of data transferred between the nodes, and the impact of this in the network infrastructure, regarding the topology and protocols on place.

For this calculation, some information must be available:

- list of tasks of all running jobs and its locations in the cluster
- list of all data blocks to being read for each running task, and its locations in the cluster
- list of all network paths in the cluster, following the network topology
- list of all nodes in the cluster and its locations in the network topology, in which rack each node is
- network features (bandwidth, latency, etc)

As shown in the algorithm, the list of tasks is evaluated, and the external and rack tasks are selected for the network traffic calculation. For each task, the paths used between the

processing node and the data node are obtained from the network topology graph, according to the switching protocol in place. All the traffic flows in all the paths are accumulated by the algorithm, and placed in a list. For this, the underling network features are take into account, like the frame size, payload size and overhead. The method to obtain the network path also takes into consideration the protocols in place, considering multiple paths and load balancing when available and requested. The analysis is executed later based on this list, following the options available in the simulator. The traffic can be considered as a simple accumulation or average, but can be also considered as a step list, when the network flow pairs are solved in order, regarding the available bandwidth in each network segment.

Algorithm 1: Network Traffic Calculation Algorithm

```
// considering i,j as processingNode and
   dataNode
input : jobTasks: List< task >, dataBlocks:
        List< block >, steps: List< path_{i,j} >
output: segmentList: List< segments >
for task ∈ jobTasks do
    if task.type ≠ LOCAL then
        // get processing and data node from
           task
        processingNode, dataNode ← getNodePair(task)
           // get network path between the
           nodes
        taskNetSegments ←
         getNetPath(processingNode, dataNode)
        // get the data blocks for the task
        dataBlocks ← getDataBlocks(task)
        for segment ∈ taskNetSegments do
            // aggregate traffic in list
            increaseTrafficSegment(segment, dataBlocks,
             segmentList)
return segmentList
```

D. Simulation Usage Examples

The main purpose of this simulator is to help in research concerning novel strategies in data and task placement in big data frameworks, and in novel network designs and tools to improve performance and reduce energy consumption in the mentioned frameworks.

In this sense, and based on the available features, there are some test scenarios better suited for the simulator, as follows:

1) Evaluation of data placement strategies on HDFS: These strategies can have an impact in how the tasks are executed, due to the need to read and aggregate the source data. The simulator provides a set of Java interfaces to implement the main methods that control the data placement, with the intention of making this implementation simpler.

2) Evaluation of task placement strategies or scheduling algorithms: These strategies and algorithms can also have an impact in performance and network usage, so the simulator was designed to test these as well. In the current state, the simulator can simulate MapReduce API over YARN, using the standard scheduler for tasks. As the class structure was prepared using the same level of abstraction of data placement strategies, more options will be implemented in the future (like Spark), besides custom strategies as well.

3) Evaluation of network topologies: The simulator has a conventional hierarchical topology implemented, with a Spine-Leaf strategy, and new ones can be implemented using graphs, which include complex topologies. At least one complex topology, based on rings and parallel paths was implemented as a evaluation of data center networks, using the simulator graph capabilities.

4) Evaluation of network protocols and routing/switching strategies: The protocols STP (Spanning Tree Protocol) and SPB (Shortest Path Bridging) are already implemented, and any cluster and topology can be tested using. Other protocols or routing/switching strategies can be implemented in a simple way, using interfaces to order behavior. At least one more was already implemented, for a particular research.

It is important to remember that the simulator implements the reading phase of big data job execution, not being suitable for simulations that involve other phases than this one.

III. Evaluation set-up

In this paper we aim at answering the following three questions:

- **RQ 1**: Is BigDataNetSim's data placement model accurate?
- **RQ 2**: Is BigDataNetSim's job placement model accurate?
- **RQ 3**: How does BigDataNetSim scale to large clusters?

We use a real cluster of 20 machines (+ 1 master nodes) to evaluate the accuracy of our models (RQ 1 & 2). Hadoop installed version is 2.7.6 in a single and dual rack network configuration. The nodes are regular PCs, with Intel Core i5 processors, 8 GB of RAM and SATA hard drives with 1 TB each and the master is a Core i7 with 32 GB of RAM.

A. Jobs executed

As the simulator is designed to mainly generate data from the reading phase of big data jobs (the reading part of Map phase, in MapReduce jobs, for instance), we focused on the kind of tasks where this particular phase is prominent. We need jobs with significant amount of data to read, in order to force a heavy network traffic in high level of concurrency situations, and as the remaining phases of the frameworks are not covered by the simulator, the tests will not cover the additional phases. The amount of data is also related with the size of the cluster, as the intention of the simulations is to use most or all of the network resources available. In this sense, we need to occupy the highest possible number of processing slots available in the cluster. Therefore, considering that a processing slot is

responsible for reading a block of data (128 MB in regular HDFS configurations), the amount of data to be processed can increase as the size of the cluster increase as well.

For these reasons, we selected 3 job types to test the simulator:

- TestDFSIO: Test job distributed with Hadoop that generates configurable amounts of data to write and read from HDFS using MapReduce jobs.
- WordCount: An standard job that counts the occurrences of words in a text data input.
- TPC-H query 6: A query based on a large reading portion and little processing running on a structure dataset.

As the TPC group is well-know for the standardized tests, most of our tests were conducted using this last one. As shown in the following listing, TPC-H Query 6 is based on a single table (thus a single CSV file in HDFS) and goes through the entire file to select part of the dataset to execute an aggregation operation.

```
1 SELECT
2     sum(l_extendedprice * l_discount) as
          revenue
3 FROM
4     lineitem
5 WHERE
6     l_shipdate >= date '1994-01-01'
7     AND l_shipdate < date '1994-01-01' +
          interval '1' year
8     AND l_discount between 0.06 - 0.01
          AND 0.06 + 0.01
9     AND l_quantity < 24;
```

As the reduce phase is limited to an aggregation function and a simple arithmetic operation, the reading part will be the majority of the time consumed, keeping the tests focused on the results the simulator can obtain, and don't spending time in operations not covered by the simulator (like the shuffle and reduce phases).

B. Metrics

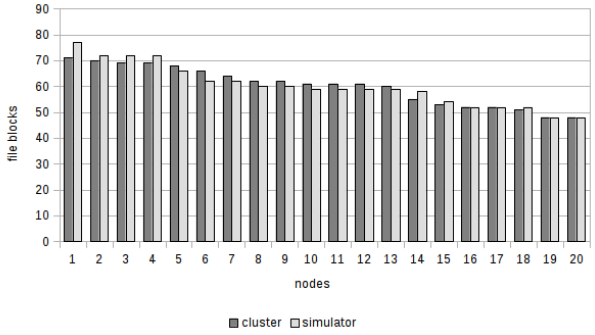
The simulator accuracy is based on two metrics, the degree of data distribution in the distributed file system, and the distribution of tasks among the local, rack and external classes.

For the data distribution metric, we measure the average number of blocks per node in the cluster, and the standard deviation of this value. As most of the choices made by the frameworks rely upon the data locality, an accurate distribution of data in the simulated file system is important in order to have the most approximate results for the job execution.

And for the task placement, the classification among local, rack or external shows how much data is transmitted over the network, and between which computers, and this can be modeled for different topologies and network features. This metric is important not only for being a measure of the overall performance of a job (as data transmitted over the network poses as a bottleneck), but also to allows a realistic analysis of the network paths used in different network topologies, using different network protocols.

Besides these two functional metrics, a performance analysis was conducted, to show how much time each simulation run can take, in order to determine the expected time to complete a longer simulation.

Fig. 6. Block distribution - 50GB.



IV. Evaluation Results

We have conduct tests using the simulator and the described test bed, and the results show that the simulator have a reasonable accuracy when compared to a real setup. We presented three ways to compare and assess the accuracy and efficiency of the simulator: data distribution, task distribution in job executions and a performance measurement of execution.

A. Data distribution in HDFS

The data distribution reports show how the blocks of a particular file are distributed among the cluster nodes. The simulation takes into account HDFS replication factor and the data placement policy.

For these tests, were considered the HDFS standard replication factor of 3, and the default block placement strategy. The tests were executed in files with 1, 3, 5, 10, 20, 30, 40 and 50 GB in the physical cluster, and in a simulation. Both clusters have 20 nodes and 2 racks. The charts shown the block distribution among the nodes, and the standard deviation for the files.

In Figure 6, we can observe the distribution of file blocks among the cluster nodes for a 50GB file. The values are ordered in descending order, to better compare the results. The distribution of blocks is similar, with a maximum difference of 7.7% and an average difference less than 1%.

Figure 7 shows the comparison of standard deviation for number of blocks per node for each file size. The standard deviation shows how even is the distribution of blocks among the nodes, and this can have an impact on how the processing engines locate the data. The standard deviation decreases as the file size increases, for both the physical and the simulated cluster, being slightly bigger in the simulations, but the progression follows the same pattern, with the difference decreasing as the file size increases. For instance, in the 1GB file, the difference in standard deviation between the physical

Fig. 7. Data distribution comparison.

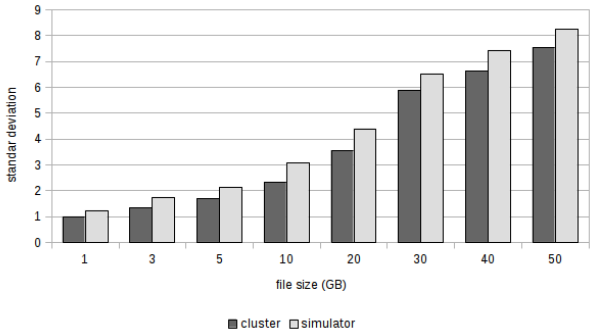
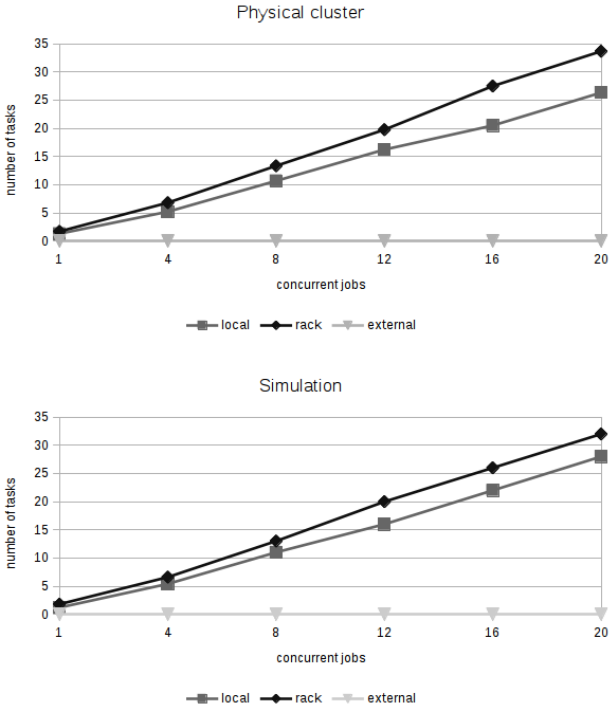


Fig. 8. Task distribution comparison.



cluster and the simulated one is around 19%, in the 50GB file is around 8%.

Judging by this results, we can conclude that the file system simulation is adequate, when compared to a real testbed, not presenting differences that could affect the results of the simulations executed on the file system.

B. Task distribution in YARN/MapReduce

As mentioned earlier, the classification of tasks between local, rack and external is important to better measure the amount of data movement in the network.

For this, the tested scenario was planned to use all the available processing slots in the cluster, determined by the configured number of vCores in each node. The tests were conducted with concurrent jobs varying from 1 to 20, in steps of 4 (1, 4, 8, 12, 16 and 20). For a physical cluster of 20 nodes with 4 vCores per node, this is enough to occupy all the slots.

Both tests (simulator and physical cluster) were configure with the Speculative Map Execution as off, and with the yarn.scheduler.capacity.node-locality-delay turned on, using double the number of nodes as metric. Those parameters affect how many tasks are launched to satisfy a given map function, and how much time YARN will wait for an ideal node with data locality guaranteed [23].

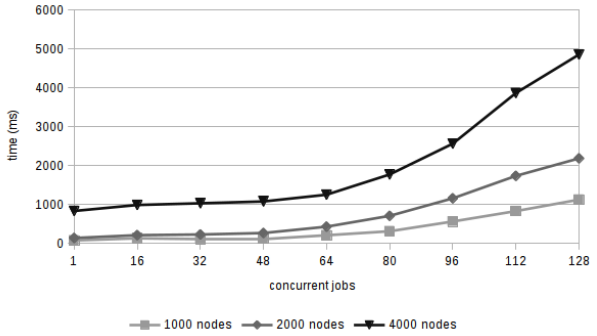
Figure 8 shows the comparison between the results in the cluster and in the simulator, showing similar lines. The average difference is 4.4%, with the difference being bigger with a small number of concurrent jobs, when the random parts of the algorithms, combined with a underutilized cluster, prevail.

C. Scalability and performance on simulations

Another important metric is the performance of the simulator itself. The amount of time spent by the simulator to run a particular test case was measured, and the results show that even for larger clusters, the performance is reasonable, even for a conventional desktop computer.

The tests were conducted in a desktop computer equipped with a Intel Core i7 processor (4 cores at 3.40GHz) with 32 GB of RAM memory, running Linux Operating System (LinuxMint v18.02) and Oracle JDK 8. All the tests results are from only one execution, but in real tests, it's expected to run multiple tries (up to 100) in order to avoid any statistical errors.

Fig. 9. Scalability analysis.



Usually, the simulator is used to execute a set of tests for a particular cluster and job configuration, running from 1 concurrent job to 128 or 256 concurrent jobs, in steps of 16 or 32. Each step is executed 100 times, and the results are aggregated. For some configurations, a test set like this can take more than 30 minutes to execute.

Figure 9 shows the results for 1000, 2000 and 4000 nodes clusters, running a number of concurrent jobs.

V. Conclusion

As shown by the experiment results, we can answer the research questions with some degree of confidence. As shown by the experiment results, BigDataNetSim is capable of simulate with accuracy the data placement algorithm of HDFS, in order to provide accurate results for the simulations on top of it. Using the abstraction in its implementation, it is possible to

implement and test novel data placement strategies and study its impact on the overall operation of a big data framework.

BigDataNetSim is also capable of accurately simulate the number of local, rack and external tasks, which is important to provide accurate data for simulations involving network topologies and protocols, besides the task placement strategy itself.

The graphical user interfaces provided are alse useful for novel network topologies design, and makes these analysis simpler and easier to evolve.

Based on its expected usage, the simulator have some limitations. As the research that originated the simulator is focused on network information and reading phase, so this limits the expected results and the coverage of the simulations. The simulations are not considering subsequent phases of processing engines, like the combiner, sort or reduce phase of MapReduce, mainly because these phases transfer less data through the network, compared to the reading phase, and because as these phases can vary in several ways (apart from MapReduce, and considering Spark, Impala, Tez, etc), making the simulation more complex. The impact is reduced by the bigger usage of the reading phase compared with the subsequent ones, specially in network transfer.

Ultimately, the simulator is useful for analysis and development of data and task placement strategies and network topologies and algorithms, for the impact on the network usage, and performance estimates.

A. Future works

As future developments, will be implemented new job processing frameworks, like Spark, and new scheduling algorithms. Some features of Hadoop 3, like the new HDFS Erasure Coding will be added as configuration options in the simulator, as these features can impact future research.

The use of an external tool like Gephi to help in the design of complex network topology will be studied. Currently, Gephi is used only to visualize the network topology, but this can be extended in the future, as well as a Mininet script generator based on Gephi files.

And finally, the development team will start efforts to make the source code available for the community.

References

[1] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu, "Mrsim: A discrete event based mapreduce simulator," in *7th Int'l Conf. on Fuzzy Systems and Knowledge Discovery*, 2010, pp. 2993–2997.

[2] A. Verma, L. Cherkasova, and R. H. Campbell, "Play it again, simmr!" in *IEEE International Conf. on Cluster Computing*, 2011, pp. 253–261.

[3] W. Kolberg, P. D. B. Marcos, J. C. S. Anjos, A. K. S. Miyazaki, C. R. Geyer, and L. B. Arantes, "Mrsg - a mapreduce simulator over simgrid," *Parallel Comput.*, vol. 39, no. 4-5, pp. 233–244, 2013.

[4] A. Mumak. (2008) Apache mumak. [Online]. Available: https://github.com/facebookarchive/hadoop-20/tree/master/src/contrib/mumak

[5] J.-C. Lin, I. C. Yu, E. B. Johnsen, and M.-C. Lee, "Abs-yarn: A formal framework for modeling hadoop yarn clusters," in *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering - Volume 9633*, 2016, pp. 49–65.

[6] F. Teng, L. Yu, and F. Magoulès, "Simmapreduce: A simulator for modeling mapreduce framework," in *5th FTRA International Conference on Multimedia and Ubiquitous Engineering*, 2011, pp. 277–282.

[7] Apache. (2013) Yarn scheduler load simulator. [Online]. Available: https://hadoop.apache.org/docs/r2.4.1/hadoop-sls/SchedulerLoadSimulator.html

[8] J. Jung and H. Kim, "Mr-cloudsim: Designing and implementing mapreduce computing model on cloudsim," in *2012 International Conference on ICT Convergence (ICTC)*, 2012, pp. 504–509.

[9] Y. Liu, M. Li, N. K. Alham, and S. Hammoud, "Hsim," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 300–308, 2013.

[10] 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 *CIDR*, vol. 11, no. 2011, 2011, pp. 261–272.

[11] Y. Qiao, X. Wang, G. Fang, and B. Lee, "Doopnet: An emulator for network performance analysis of hadoop clusters using docker and mininet," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, 2016, pp. 784–790.

[12] D. Ardagna, E. Barbierato, A. Evangelinou, E. Gianniti, M. Gribaudo, T. B. M. Pinto, A. Guimarães, A. P. Couto da Silva, and J. M. Almeida, "Performance prediction of cloud-based big data applications," in *ICPE, year = 2018, pages = 192–199*.

[13] R. Panda, X. Zheng, A. Gerstlauer, and L. K. John, "Camp: Accurate modeling of core and memory locality for proxy generation of big-data applications," in *DATE*. IEEE, 2018, pp. 337–342.

[14] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[15] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in hadoop mapreduce," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2014–2015, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367562

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE 26th Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10.

[17] C. Szyperski, M. Petitclerc, and R. Barga, "Three experts on big data engineering," *IEEE Software*, vol. 33, no. 2, pp. 68–72, 2016.

[18] Randstad. (2017) Five best it jobs for 2017. [Online]. Available: https://www.randstadusa.com/jobs/career-resources/in-demand-jobs/best-in-demand-information-technology-jobs/

[19] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The end of an architectural era: (it's time for a complete rewrite)," in *VLDB*, 2007, pp. 1150–1160.

[20] A. Giraph. (2012) Apache giraph. [Online]. Available: http://giraph.apache.org/

[21] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *SIGMOD*, 2010, pp. 135–146.

[22] F. G. Villanustre, "Big data trends and evolution: A human perspective," in *Proceedings of the 3rd Annual Conference on Research in Information Technology*, 2014, pp. 1–2.

[23] A. Hadoop. (2008) Hadoop documentation. [Online]. Available: http://hadoop.apache.org/docs/current/

[24] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154