

This is a postprint version of the following published document:

Caíno-Lores, S., Fernández, A. G., García-Carballeira, F. & Pérez, J. C. (2015). A cloudification methodology for multidimensional analysis: Implementation and application to a railway power simulator. *Simulation Modelling Practice and Theory*, 55, 46–62.


DOI: [10.1016/j.simpat.2015.04.002](https://doi.org/10.1016/j.simpat.2015.04.002)

© 2015 Elsevier B.V. All rights reserved.



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

# **A cloudification methodology for multidimensional analysis: Implementation and application to a railway power simulator**

Silvina Caíno-Lores , Alberto García Fernández, Félix García-Carballeira, Jesús Carretero Pérez

Computer Science and Engineering Department, University Carlos III of Madrid, Avda. Universidad 30, 28911 Leganes, Madrid, Spain

## **Abstract**

Many scientific areas make extensive use of computer simulations to study complex real-world processes. These computations are typically very resource-intensive and present scalability issues as experiments get larger even in dedicated clusters, since these are limited by their own hardware resources. Cloud computing raises as an option to move forward into the ideal unlimited scalability by providing virtually infinite resources, yet applications must be adapted to this new paradigm. This process of converting and/or migrating an application and its data in order to make use of cloud computing is sometimes known as cloudifying the application. We propose a generalist cloudification method based in the MapReduce paradigm to migrate scientific simulations into the cloud to provide greater scalability. We analysed its viability by applying it to a real-world railway power consumption simulator and running the resulting implementation on Hadoop YARN over Amazon EC2. Our tests show that the cloudified application is highly scalable and there is still a large margin to improve the theoretical model and its implementations, and also to extend it to a wider range of simulations. We also propose and evaluate a multidimensional analysis tool based on the cloudified application. It generates, executes and evaluates several experiments in parallel, for the same simulation kernel. The results we obtained indicate that our methodology is suitable for resource intensive simulations and multidimensional analysis, as it improves infrastructure's utilization, efficiency and scalability when running many complex experiments.

## **1. Introduction**

Scientific simulations constitute a major set of applications that attempt to reproduce real-world phenomena in a wide range of areas such as engineering, physics, mathematics and biology. Their complexity usually yields a significant resource usage regarding CPU, memory, I/O or a combination of them.

In order to properly scale these applications, they can be distributed to a cluster or grid. While these approaches have proved successful, they often rely on heavy hardware investment and they are tightly conditioned by the available resources. This de facto limits actual scalability and the addressable simulation size. Since sharing resources across multiple clusters implies several limitations, cluster applications cannot be considered sustainable, because their scalability is strongly dependent on the cluster size.

Despite scientific simulations will likely benefit from the upcoming Exascale infrastructures [1], the challenges that must be overcome –power consumption, processing speed and data

locality, for instance [2]– will probably rise again in the future as applications become more complex. Therefore, the ideal situation of unlimited scalability seems difficult to reach with this approach.

Another option is cloud computing, which has been increasingly studied as an alternative to traditional grid and high-performance distributed environments for resource-demanding and data-intensive scientific simulations [3]. Cloud computing emerged with the idea of elasticity: virtual unlimited resources obtainable on-demand with minimal management effort [4]. It would enable the execution of large simulations with virtual hardware properly tailored to fit specific use cases like memory-bound simulations, CPU-dependent computations or data-intensive analysis. It holds further advantages, such as elasticity, automatic scalability and instance resource selectivity. Moreover, its so-called pay-as-you-go model allows to adjust the required instances to the particular test case size while cutting-down the resulting costs.

Furthermore, recent advances in cloud interoperability and cloud federations can contribute to separate application scalability from datacenter size [5,6]. From that point of view, applications would become more sustainable, as they can be operated in a more flexible way through heterogeneous hardware, cross-domain interactions, and shared infrastructures. There are several issues that should be tackled in order to develop a sustainable application, such as:

- Virtual unlimited scalability could be achieved by reducing the number architectural bottlenecks, such as network communications or master-node dependencies. This would minimize the added overhead of working with more nodes, making a better use of the available resources.
- By making the application platform independent, we can aggregate computational resources possibly located in different places, hence local data center size would not be a limitation. Moreover, we can exploit cluster and cloud resources simultaneously following an hybrid scheme.
- An application that could behave in a flexible manner efficiently would be able to scale up or down easily according to instantaneous user needs, thus adapting computing resources to specific simulation sizes and deadlines.
- If the application already exists and has to be adapted, it is desirable to minimize the impact on the original code, thus performing the minimal modifications needed to achieve the aforementioned objectives.

Given the former, resource-intensive scientific simulations hold potential scalability issues on large cases, since standalone and cluster hardware may not satisfy simulation requirements under such stress circumstances. Therefore, in previous work we have explored the possibility of performing a paradigm shift from single-node HPC computations to a datacentric model that would distribute the simulation load across a set of virtual instances [7,8]. In this paper we propose a generic methodology to transform scientific simulations into a cloud-suitable data-centric scheme via the MapReduce framework. Moreover, in this paper we provide an optional experiment generation stage that allows users to configure a full set of simulations with a varying

parameter for solution optimization purposes. This multidimensional analysis capability is translated to a many-task problem within our methodology.

The processes mentioned are illustrated by means of a real-world application, a simulator which calculates power consumption on railway installations. This simulator, starting from the train movements (train position and consumption), calculates the instantaneous power demand (taking into account all railway elements such as tracks, overhead lines, and external consumers) indicating whether the power provisioned by power stations is enough or not. Simulator internals consist on composing the electric circuit on each instant, and solving that circuit using modified nodal analysis. The starting version of the simulator, based on multi-threading, is memory bounded, strongly limited by the number of instants to be simulated simultaneously (and therefore by the number of threads). The resulting performance is evaluated on Amazon Elastic Compute Cloud running Hadoop YARN MapReduce.

The rest of this paper is organized as follows: Section 2 discusses related works, Section 3 describes our proposed methodology, Section 4 illustrates the cloudification transformation method on a particular use case, Section 5 evaluates how the resulting design implementation on Hadoop MapReduce 1.1.2 (MRv1) and Hadoop YARN Mapreduce 2.2.0 (MRv2) behaves on both a local cluster and Amazon Elastic Compute Cloud (EC2), Section 6 describes the process of transforming the methodology into a multidimensional analysis by means of a many-task experiment generation and evaluation process and, finally, Section 8 provides key ideas as conclusions and some insight in future work.

## 2. Related work

Scientific applications and their adaptability to new computing paradigms have been dragging increasing attention from the scientific community in the last few years. The applicability of the MapReduce scheme for scientific analysis has been notably studied, specially for data-intensive applications, resulting in an overall increased scalability for large data sets, even for tightly coupled applications [9].

Hadoop MapReduce is nowadays widely used as base platform for new programming languages and architectures. Hadoop MapReduce is used in Pig Latin [10], an associative language used in Yahoo for taking advantage of both declarative languages and map-reduce programming style. This approach is strongly focused on processing data sets, and does not tackle the issue of scientific workflows. Apache Hive [11] and Bigtable [12] are two storage systems developed on the top of Hadoop. Hive expresses data queries in an SQL-like declarative language which is compiled into map-reduce jobs. Bigtable uses a sparse, distributed, multi-dimensional sorted map to provide a fast method to access data, although the data structures exposed to the user are rows, columns, and tables.

Nevertheless, the most popular evolution of Hadoop MapReduce is Spark [13]. Spark evolves the map-reduce programming style operating on resilient in-memory distributed data sets, thus improving performance in workflows (since the data does not have to be written to disk between tasks). Finally, an approach more related to scientific simulations is Twister [14], a runtime

specifically designed for iterative map-reduce works and, therefore, more suitable for repetitive computations over the same data.

The possibility to run simulations in the cloud in terms of cost and performance was studied in [15]. This work concludes that performance in the Abe HPC cluster and Amazon EC2 is similar – besides the virtualization overhead and high-speed connectivity loss in the cloud –, and that clouds are a viable alternative for scientific applications. Hill [16] investigated the trade-off between the resulting performance and achieved scalability on the cloud versus commodity clusters. Despite at the time of this work the cloud could not properly compete against HPC clusters, its low maintenance and cost made it a viable option for small scale clusters with a minimum performance loss.

The relationship between Apache Hadoop MapReduce and the cloud for scientific applications has also been tackled in [17], which establishes that performance and scalability tests results are similar between traditional clusters and virtualized infrastructures. In [18], Srirama, Jakovits and Vainikko study how some scientific algorithms could be adapted to the cloud by means of the Hadoop MapReduce framework. They establish a classification of algorithms according to the structure of the MapReduce schema these would be transformed to. They suggest that not all of them would be optimally adapted by their selected MapReduce implementation, yet they would suit other similar platforms such as Twister or Spark. They focus on the transformation of particular algorithms to MapReduce by redesigning the algorithms themselves, and not by wrapping them into a cloudification framework as we propose. A similar approach is HAMA [19], a framework which provides matrix and graph computation primitives on the top of MapReduce. An advantage of this framework over traditional MPI approaches to matrix computations is the fault tolerance provided by the underlying Hadoop framework. Finally, an approach for using Hadoop MapReduce in scientific workflows is that explained in [20], whose authors propose a new architecture named SciFlow. This architecture consists on a new layer added on the top of Hadoop, enhancing the patterns exposed by the framework with new operations (join, merge, etc.). Scientific workflows are represented as a DAG composed of these operations.

More related with the proposed approach is the so-called parameter sweep [21], in which the same simulation kernel is executed multiple times with different input parameters, thus providing task independence. A related point of view is the many-task computing paradigm [22], in which a high number of loosely coupled tasks are executed over data sets for a short time. In this context, cloud computing has been proved as a good solution for scientists who need resources instantly and temporarily for fulfilling their computing needs [23]. On the other hand, these evaluations show that better performance is needed for clouds to be useful for daily scientific computing. SciCumulus [24] is proposed as a lightweight cloud middleware to explore many-task computing paradigm in scientific workflows. This middleware includes a desktop layer to bring the scientist the possibility of composing their own workflows, a distribution layer to schedule the flows in a distributed environment, and an execution layer to manage the parallel execution of the tasks. The preliminary results demonstrate the viability of the architecture.

In this context, trends are naturally evolving to migrate applications to the cloud by means of several techniques, and this includes scientific simulations as well. D’Angelo [25] describes a

Simulation-as-a-Service schema in which parallel and distributed simulations could be executed transparently, which requires dealing with model partitioning, data distribution and synchronization. He concludes that the potential challenges concerning hardware, performance, usability and cost that could arise could be overcome and optimized with the proper simulation model partitioning. Application cloudification middlewares have also been developed to allow legacy code migration to the cloud. For instance, in [26] a virtualization architecture is implemented by means of a Web interface and a Software-as-a-Service market and development platform. This generalist approach is suitable to provide multi-tenancy in desktop applications, but might not suffice for the resource-intensive computations required by large-scale simulations.

Finally, in [27] we find interesting efforts to move desktop simulation applications to the cloud via virtualized bundled images. These run in a transparent multi-tenant fashion from the end user's point of view, while minimizing costs. As previously discussed, we believe the virtualization middleware might affect performance since it does not take into account any structural characteristics of the model, which could be exploited to minimize cloudification effects or drastically affect execution times or resource consumption.

### 3. Methodology description

The MapReduce paradigm consists of two user-defined operations: map and reduce. The former takes the input and produces a set of intermediate  $\langle \text{key}; \text{value} \rangle$  pairs that will be organized by key by the framework, so that every reducer gets a set of values that correspond to a particular key [28].

As a data-centric paradigm, in which large amounts of information can be potentially processed, these operations run independently and only rely upon the input data they are fed with. Thus, several instances can run simultaneously with no further interdependence. Moreover, data can be spread across as many nodes as needed to deal with scalability issues.

Simulations, however, are usually resource-intensive in terms of CPU or memory usage, so their scalability is limited to hardware restrictions, even in large clusters. Our goal is to exploit the data-centric paradigm to achieve a virtually infinite scalability. This would permit the execution of large numeric simulations independently of the underlying hardware resources, with minimal effects to the original simulation code. From this point of view, numeric simulations would become more sustainable, allowing us to spread simulation scenarios of different sizes in a more flexible way, using heterogeneous hardware, and taking advantage of shared inter-domain infrastructures.

To achieve this, we will take advantage of MapReduce's lack of task interdependence and data-centric design. This will allow to disseminate the simulation's original input to distribute its load among the available nodes, which will yield the scalability we aim for. The steps involved in our proposed methodology are described in the following sections.

### 3.1. Application analysis

Our purpose is to divide the application into smaller simulations that can run with the same simulation kernel, but on a fragment of the full partitioned data set, so that we can parallelise the executions and lower the hardware requirements for each.

Hence, we must analyse the original simulation domain in order to find an independent variable  $-T_x$  in Fig. 1– that can act as index for the partitioned input data and the following procedures. This independent variable would be present either in the input data or the simulation parameters and it could represent, for example, independent time-domain steps, spatial divisions or a range of simulation parameters.

At the moment, the analysis and selection of such variable is done by direct examination of the original application. As this process is critical, and it is intimately related with the application's structure, procedures and input, it should be performed by an expert in the simulator to be cloudified. Future work could simplify this stage by mean of automatic analysis and variable proposal, yet an expert would still be needed to assess the correctness of the suggestion.

### 3.2. Cloudification process design

Once verified that the application is suitable for the process, it can be transformed by matching the input data and independent variables with the elements in Fig. 1. This results in the two MapReduce jobs described below:

Adaptation stage: reads the input files in the map phase and indexes all the necessary parameters by  $T_x$  for every execution as intermediate output. The original data must be partitioned so that subsequent simulations can run autonomously with all the necessary data centralized in a unique  $\delta T_x; parametersP$  entry.

Simulation stage: runs the simulation kernel for each value of the independent variable along with the necessary data that was mapped to them in the previous stage, plus the required simulation parameters that are shared by every partition. Since simulations might generate several output files, mappers would organize the output by means of file identifier numbers as keys, so as reducers could be able to gather all the output and provide final results as the original application.

### 3.3. Virtual cluster planning

The former stages would most likely require different amounts of CPU and memory resources, depending on the application to be cloudified. In this section, we provide an heuristic to detect the slaves' instance requirements to maximize resource utilization. First of all, we define the concept of an entry. An entry is a piece of data processed by the mappers on any of the methodology stages. There will be different entries for the adaptation and simulation phases:



For the adaptation phase, an entry is one of the registers by which the input files are arranged according to. Mappers in adaptation phase process these registers indexing them according the independent variable  $T_x$ .

For the simulation phase, an entry is one of the  $\delta T_x$ ; parameters  $P$  pair. This is the autonomous piece of data which is going to be simulated by the mappers of the simulation phase.

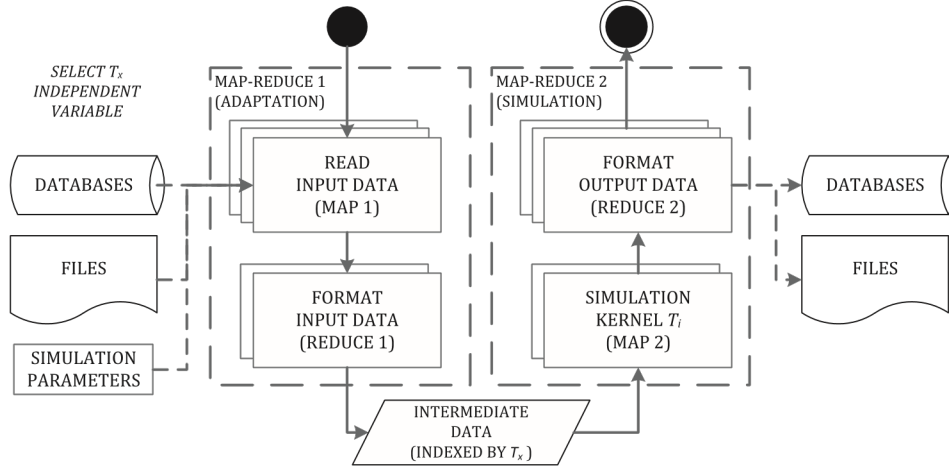


Fig. 1. Methodology overview.

Next, we consider the following assumptions:

All the slaves must be equal in terms of memory and number of cores.

The execution time required to process an entry,  $t_e \in \mathbb{R}^+$ , is known and homogeneous for all the entries. For the adaptation phase, this is the time required to parse an index the parameters according to the independent variable. For the simulation phase, this is the time required to execute the simulation kernel over a single autonomous piece of the input data.

The amount of memory required to process an entry,  $m_e \in \mathbb{R}^+$ , is known and homogeneous for all the entries. For the adaptation phase, this is the memory needed to perform the grouping of input parameters according to the independent variable. For the simulation phase, this is the memory needed to allocate and simulate one single simulation kernel and merge the results.

The number of entries,  $n_e \in \mathbb{N}$ , is known.

Of course,  $t_e$  and  $m_e$  may vary, depending on the problem domain. In that case the expert should either estimate an accurate value, or consider inserting probabilistic distributions in formulae. In this first approach, we assume  $t_e$  and  $m_e$  constant along all entries, in order to simplify the heuristic. Given these parameters and assumptions, the objective is to minimize the total execution time of the cloudified application,  $T \in \mathbb{R}^+$ . The minimization problem is defined as follows:

$$\min T = ((t_e * n_e) / (n_i * C_i)) + a$$



where  $c_i \in \mathbb{N}$  represents the number of cores per instance,  $n_i \in \mathbb{N}$  is the number of instances in the targeted cluster, and  $a \in \mathbb{R}^+$  is a parameter that represents the compute overhead factor of the underlying platform (spawning the tasks, etc.), which is considered constant.

Subject to the following constraint:

$$m_e * C_i + b \leq m_l \quad (2b)$$

where  $m_l \in \mathbb{R}^+$  represents the amount of memory per instance and  $b$  represents the memory overhead added by the platform on each instance. Eq. 2 indicates that the aggregated memory required by the entries that can be concurrently processed by each node must not exceed the total memory of it.

Once  $n_i$ ,  $c_i$  and  $m_l$  are found, one can select the instances that have greater or equal resources for both metrics, simultaneously.

This minimization problem can be modified by letting  $T$  be a fixed value, in order to find suitable instances to meet a specific deadline. This deadline-oriented planning can be very beneficial to minimize costs in pay-as-you-go infrastructures, as deadlines can be multiples of the time slices covered by successive charges. For instance, Amazon charges the user for slices of one hour (even if the user only allocates VMs for 15 min). Therefore, this heuristic can be used for: (a) providing a VM configuration and calculate the expected execution time using that configuration, or (b) choosing the best VM configuration in order to meet a given deadline.

As an example of the former formulation, let  $E$  be the set of  $n_e = 10^5$  entries the user wants to process, each one requiring  $m_e = 1$  GB of memory and  $t_e = 1$  s for its execution. Assuming the user wants to process  $E$  within one hour, using instances with  $c_i = 2$  and  $m_l = 17$  GB, the execution time,  $T$ , becomes a deadline of 3600 s. Let assume  $a = 30$  s and  $b = 1$  GB. Solving the resulting equation, the number of instances necessary to process all entries within one hour is 15.

#### 4. Case study

To illustrate how this methodology works on a real-world use case, we applied it to transform a memory-bound railway electric power consumption simulation. We selected this tool due several facts. First of all, it is a real tool currently used by ADIF, the Spanish railway company, to test and verify different scenarios (e.g. developing new routes, increasing train traffic across the tracks, or testing failure situations where services have to be operated on degraded mode), so it portrays a general sort of engineering simulators commonly used which would be desirable to move to the cloud. Secondly, the tool requires a high amount of computing power, performing multiple matrix operations for each simulated instant (and a typical train traffic scenario has to be simulated during the whole day), so it is worthwhile improving the application scalability in order to reduce simulation times. Thirdly, the application is memory-bound, as will be explained in Section 4.2, so it is a perfect test case for our methodology.

#### 4.1. Application description

The aim of this simulator is, provided a number of trains circulating across the lines, calculate if the amount of power supplied by the electrical substations is enough or not. Starting from a description of the railway infrastructure (i.e. tracks, catenaries deployed over the tracks, electric substations placed along the tracks, as well as additional elements like feeders and switches, the simulator reads the position of the trains and its instantaneous power demand. Then, the electric circuit formed by the trains and the infrastructure is composed and solved using modified nodal analysis (MNA). Useful mean voltages, voltage drops, and temperatures of the wires are examples of results provided by the tool. The structure of the selected application is shown in [Fig. 2](#). It consists of a preparation phase in which all the required input data is read and fragmented to be executed in a predefined number of threads. Two classes of input files are handled:

- A common infrastructure specification file containing the initial and final time of the simulation, besides a wide range of domain-specific simulation parameters such as station and railway specifications and power supply definition.
- A set of train movement parameters files, structured in a time-based manner, in which each line contains the calculation of speed and distance profiles for a particular train at a specific instant regarding the infrastructure constraints, with a one second interval.

Each of the resulting threads then performs the actual simulation by means of an electric iterative algorithm, storing in shared memory the results that will be merged in the main thread to constitute the final output files. Simulator internals consist on composing the electric circuit on each instant, and solving that circuit using modified nodal analysis performing the following steps:

1. Given infrastructure data, and train positions at current instant, the matrices representing the electric circuit are composed following the MNA technique.
2. The main matrix is inverted using LU decomposition.
3. Given train consumption at present instant, the aim is to obtain the corresponding values of current and voltage (both unknown). An iterative process is conducted, performing the following substeps:

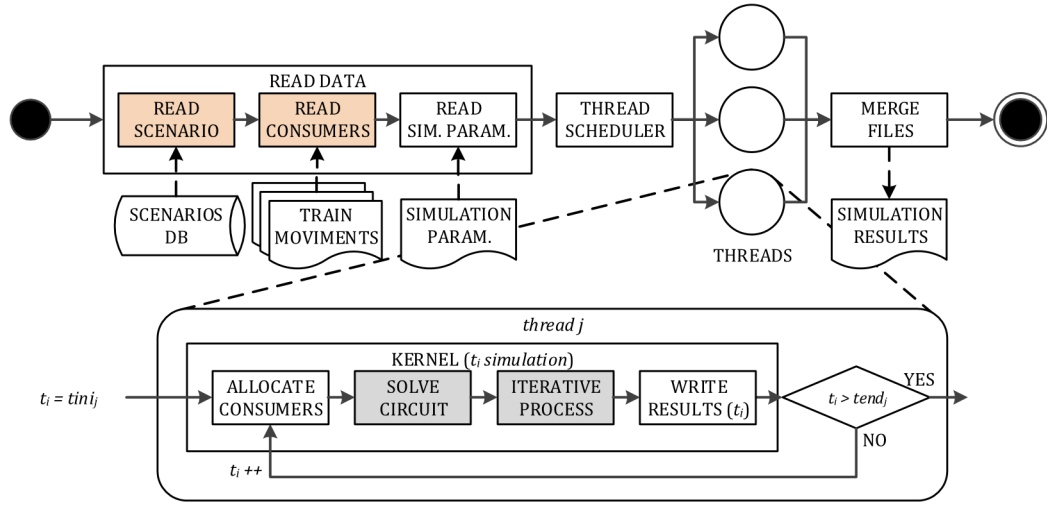


Fig. 2. Case study original simulation structure.

- (a) A value for train voltage is proposed (e.g. 3000 V).
  - (b) The circuit is solved using that proposed value. The current is obtained.
  - (c) The new consumption is calculated as the product of proposed voltage and current obtained. This new value is compared against the original one provided as input data.
  - (d) If the error is less than certain percentage (e.g. 0.5%), current and voltage values for each train have been found, so the algorithm ends. If not, another iteration is conducted, proposing a different voltage.
4. Finally, results are written to the disk.

The application is multi-threaded, so simulation workload is split among the available cores in the computer. Each thread simulates a different subset of the total simulated time (see Fig. 2).

### Resource analysis

As we said before, the application is compute and memory-bound, because its memory usage pattern leads to a lack of scalability if we want to simulate bigger and bigger problems. There are two independent factors that have influence on application memory usage, and should be analyzed independently:

**Problem size.** The number of simulated elements on the same instant (e.g. trains, tracks, catenaries, etc.) is proportional to the size of the circuit to be calculated (problem size). Actually, for each new element, a minimum of two nodes and one branch are added to the circuit (the twice as much for certain elements such as trains). Following the MNA technique, this means two more rows and one more column in the problem matrix.

**Number of simulated instants.** The circuit must be solved for each instant of the simulation.

As said before, a typical train traffic scenario has to be simulated during the whole day,

leading to 86,400 instants with the duration of one second. While we can split this workload across the node cores using threads, for each thread solving an instant the matrices must be allocated in memory. This increases the memory usage linearly to the number of threads. There is a trade-off between execution time and memory usage. The more threads we add to shorten the simulation time, the more memory we consume.

The application is also very demanding in terms of processing power, since it deals with several matrix operations per simulated instant: LU decomposition, inversion, multiplication, etc. which traditionally require a great amount of floating point operations. But unlike the memory, this limitation can be addressed either by adding more cores to a compute node, or extending the execution time of the simulation.

In order to illustrate this analysis, we conducted a study of the application memory consumption given different problem and simulation sizes. Four test cases were considered with variations on the circuit size, simulation's initial and final time and, consequently, input data volume, execution time, and memory consumption. A description of these simulations is provided in [Table 1](#). Cases I and II should not yield any significant load, yet simulation III is expected to reflect the system's behavior under average problems. The biggest experiment, case IV, should reveal the platforms' actual limitations as simulations become larger, if any. These tests are meant to indicate the performance of the cloudified adaptation versus the original application under an increasing amount of input data and simulation time. All test cases are based on the same real case, a particular railway line at Madrid surroundings, with increasingly levels of detail and simulation periods. This line has been used before in other works [\[29\]](#) because it is a good example in size and complexity of a real railway project.

[Fig. 3](#) displays the memory consumption of the application as we increase the number of concurrent threads. Measurements have been taken by means of the Linux `proc`. OS policies about memory pages assignment introduce a slight randomness, so we have repeated each measurement 10 times. As we can see in the figure, this application does not scale well for large test cases in terms of memory usage in a standalone environment (memory tests were conducted on a single node with 48 cores and 110 GB of RAM). The most determining factor is the number of simultaneous threads, which increases the memory consumption linearly. As we said before, each thread operates a different matrix which can reach a size of

Table 1 Test cases definition.

Experiment	Avg elements per instant	Simulated time (hours)	Input size (MB)
I	77	1	1.7
II	179	33	170
III	525	177	1228.8
IV	755	224	5324.8

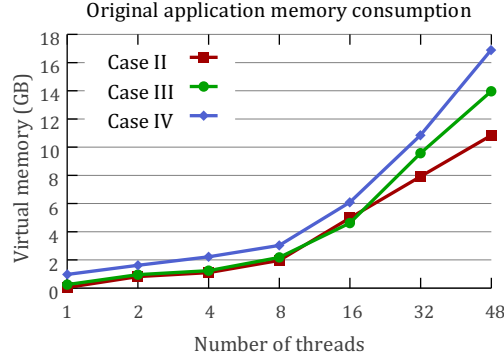


Fig. 3. Original simulation kernel memory consumption.

thousands of elements. On the other hand, using more threads is the best option to shorten the simulation time. We believe we can achieve greater scalability by cloudifying the application, since we can distribute the simulation load across several nodes. It would also disperse memory usage so that we could always add a new node in case we need to tackle a larger case. To show its feasibility, next we will apply the method described in Section 3.

#### 4.2. Cloudification

The key to adapt such algorithm to a cloud environment resides its input files, for they hold an indexed structure that stores in each line an  $\delta$ instant; parameters $\mathcal{P}$  pair. As we said before, each simulated instant ( $t_i$  in Fig. 2) is independent from the others, because for each instant the circuit has to be composed, solved, and the results obtained, so we can divide the whole simulation period (e.g. from 06:00:00 to 22:00:00) in multiple smaller simulations, each one of length 1 s. Therefore, we can consider the temporal key as the independent variable required for the theoretical model. But in order to do that, first we have to adapt the input data, rearranging that data from the initial set to multiple smaller subsets, each one containing those information necessary to simulate one single instant. Following the cloudification schema, the application was transformed into two independent MapReduce jobs executed sequentially.

In the first job, which matches the first MapReduce in Fig. 1, the movement input files,  $I_k$ , are divided into input splits by the framework according to its configuration. Each split is then assigned to a mapper, which reads each line and emits  $\delta$ key;value $\mathcal{P}$  pairs where the key is the instant,  $t_i$ , and the value is the corresponding set of parameters for such instant. The intention behind this is to provide reducers with a list of movement parameters per instant  $I_n; \dots; I_m$ —each element representing the movement of one of the trains involved in the overall system for a particular  $t_i$ —to concatenate and write to the output files, so that the simulation kernel can be executed once per instant with all the required data.

As described in Fig. 1, the output of the previous job is used as input to the mapper tasks by parsing each line. Then, the resulting data—which corresponds to the instant being processed—is passed to the electric algorithm itself along with the scenario information obtained from the infrastructure file that is also read by the mapper. The mappers' output is compound by an output file identifier  $F_j$  as key and the actual content as value.

Reducers simply act as mergers gathering and concatenating mappers’ output organized by file identifier and instant as a secondary key injected in the value content. This arranges the algorithm’s output so that the full simulation results are shown as in the original application, in which each output file contains the results for the whole temporal interval of the simulation.

#### 4.3. Implementation and platform configuration

The previous design could be implemented in any of the available MapReduce frameworks. Among them, we selected Apache Hadoop platform [30] given its popularity and community support. Its distributed file system is a great addition to the framework, since it allows automatic load balance. Moreover, it includes a distributed cache that supports auxiliary read-only file storage for tasks among all nodes, which suits neatly the shared infrastructure parameter file’s needs.

Besides the former technical features, Hadoop has been adopted into many Cloud environments, along with other MapReduce frameworks, resulting in reduced costs given its parallelism exploitation capabilities [31].

We implemented this design via Hadoop Pipes API, since the original code was written in C++ and we wanted to maximize code re-use. Despite Pipes does not allow to take full advantage of Hadoop’s potential given its limited functionality, it provided all the necessary tools to execute our framework, including map and reduce interfaces, basic data type support, and Distributed Cache access on job submission.

As a constantly changing technology, Hadoop evolved fast into more sophisticated and flexible versions. We started our implementation and tests with the stable version 1.1.2 and moved to Hadoop 2.2.0 in order to evaluate the effect of next generation MapReduce over YARN (MRv2) in terms of resource management and overall performance. The following paragraphs give an overview of both platforms configuration settings.

Table 2. Job-specific configurations on MRv1.

	Job 1	Job 2
Maximum no. of map slots (GB)	16	6
Number of reducers	2	4
JVM memory (GB)	4	8

Table 3 Platform configuration parameters for MRv2.

	Single-node cluster	Virtual cluster on EC2
--	---------------------	------------------------

Node memory (GB)	92	16
Virtual cores	16	4
Minimum allocation (GB)	1	0.5
Virtual memory ratio	4	8

Table 4. Job-specific configurations on MRv2.

	Single-node cluster		Virtual cluster on EC2	
	Job 1	Job 2	Job 1	Job 2
Container memory (GB)	1.5	7	1	6
Number of reducers	2	13	5	10

Hadoop 1.1.2 (MRV1) MapReduce’s architecture is based on a fixed number of slots that can be assigned to mappers and reducers. These are fully managed across the nodes by a unique JobTracker, which also assigns TaskTrackers, coordinates mappers and reducers and provides progress information to the client. This constitutes a single-point of failure and may become a bottleneck in very large clusters [32].

Hadoop’s Distributed File System (HDFS) replication was disabled in order to make HDFS interactions less time-consuming. The parameters shown in Table 2 permitted to achieve a significant balance between memory consumption –especially in the second job– and the required time to finish the job in the worst case tested. Besides the former, we forced reducers to wait for at least the 85% of the mappers to finish before start processing their output. This was considered to minimize the shuffle overload and maximize the available resources at the map phase, which is especially relevant in the second job.

Hadoop 2.2.0 (MRv2 on YARN), Next Generation MapReduce, encapsulates cluster resource management capabilities into YARN (Yet Another Resource Negotiator), leaving MapReduce-specific functionalities and configuration in an independent module. This avoids some scalability issues originated in the JobTracker by dividing its functionality and providing a general-purpose platform for other paradigms [33]. The MapReduce functionalities handled by the previous JobTracker were moved to the new ApplicationMaster. A ResourceManager is in charge of the cluster’s resource management and a HistoryServer provides clients with information on completed jobs. TaskTrackers were replaced with NodeManagers that are responsible for the resources and container management on each node. Each container can hold a map or reduce task and can be configured regarding the available computational power, memory and input/output capabilities of the node, which yields an increased flexibility.



Given the eager algorithm in terms of memory we are working with, our container configuration, shown in Table 4, aimed to provide enough memory for the second job in the worst case tested. With similar purposes, we implemented the configuration for node resource dedication that can be seen in Table 3. We did also maintain the slow-start configuration in both phases and the HDFS parameters mentioned previously.

## 5. Evaluation of cloudified application

In order to assess the application's performance we compared its execution times on both a cluster and the cloud. The following sections describe the utilized resources and a discussion on the obtained outcome.

### 5.1. Execution environments

Table 6 summarizes the infrastructures and software platforms on which the tests were conducted. In a first place, we tested the original multi-thread application's memory consumption and performance on a cluster node consisting of a 48 Xeon E7 cores and 110 GB of RAM (Configuration 1).

This node was also used to test the resulting cloudified application to avoid variations that may arise from heterogeneous configuration, resource differences, or network latency in case of the MapReduce application [15]. This isolation favors the multi-thread application, which is especially designed to perform in standalone environments. However, it allows to focus on the actual limiting factors that may affect scalability in large test cases like I/O, memory consumption and CPU usage. Both Hadoop versions –MRv1 and MRv2– were installed and configured on the single-node cluster to benchmark their performance against the original application (Configurations 2 and 3, respectively).

MRv2 was chosen to be deployed on EC2 given its improved resource management options and better overall performance (Configuration 4). The selected cloud infrastructure consisted of a general purpose m1.medium node as dedicated master and several memory optimized m2.xlarge machines as slaves. Table 5 shows the main aspects of the selected instances, which were selected in order to maximize the number of cores for the first stage, while holding enough memory to execute as many containers as cores with sufficient memory for the second stage. The number of slaves was selected to match roughly the resources present in Configuration 1, so that the comparison is fair for both infrastructures, leading to a total of 24 slaves to match the 48 cores present in Configuration 1. Additionally, further tests have been conducted on EC2 using a variable number of slaves, in order to check if scalability issues arise as the number of nodes increases.

### 5.2. Results discussion

As we already discussed in Section 4, the original multi-thread application's memory usage suggests a lack of scalability in a cluster environment. We will now analyse whether the cloudified

simulation behaves as expected in relation to performance and scalability by examining its execution times on several execution environments, which are shown in Fig. 4. This figure shows the time measurements obtained on the configurations in Table 6, in which the EC2 cluster is constituted

Table 5 EC2 instances description.

Type	Role	Virtual CPUs	Memory (GB)	Local storage (GB)
m1.medium	master	1	3.75	410
m2.xlarge	slave	2	17.1	420

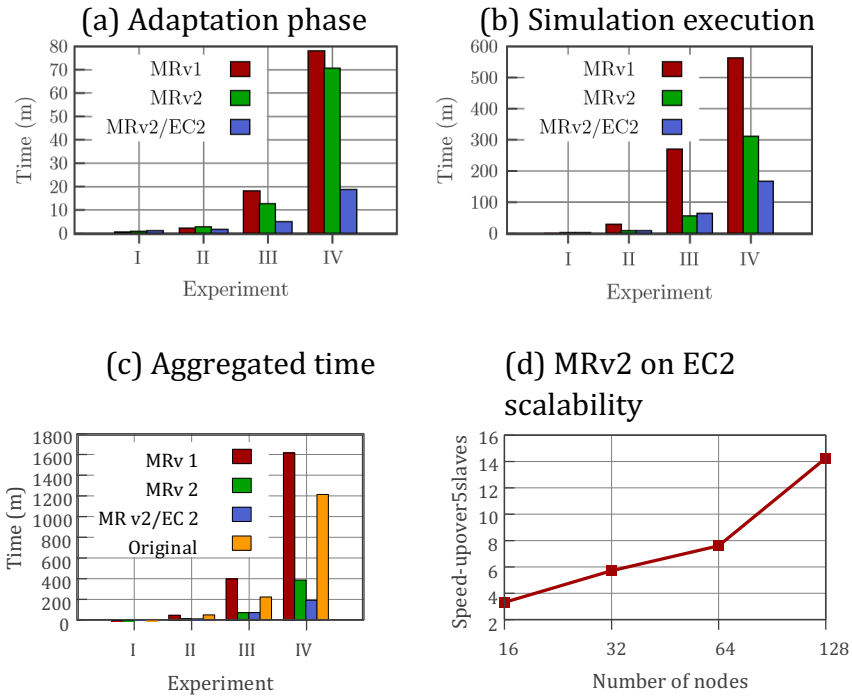


Fig. 4. Evaluation results.

Table 6. Execution environments.

Configuration	Platform	Underlying infrastructure
1	Multi-thread	Cluster node
2	Hadoop 1.1.2 (MRv1)	Cluster node
3	Hadoop 2.2.0 (MRv2)	Cluster node
4	Hadoop 2.2.0 (MRv2)	EC2

Table 7 Execution times per stage for the configurations defined in Table 6, in minutes.

Configuration	Experiment	Copy to HDFS	Adaptation	Simulation	Aggregated
MRv1	I	2.17	0.43	0.53	3.13
	II	17	2	28	47
	III	110	18	270	398
	IV	977	78	561	1616
MRv2	I	0.05	0.63	0.88	1.57
	II	0.18	2.62	8.23	11.03
	III	1.43	12.43	55.5	69.37
	IV	5.68	70.52	310.27	386.47
MRv2/EC2	I	0.12	1	0.95	2.07
	II	0.4	1.45	7.15	9
	III	1.28	4.87	63.7	69.85
	IV	5.53	18.63	165.57	189.73
Original	I	—	—	—	0.15
	II	—	—	—	46.33
	III	—	—	—	221.45
	IV	—	—	—	1221.17

by five slaves –graphs (a), (b) and (c)–. The EC2 values also served as baseline for the scalability study shown in (d). Additionally, we include the execution times in [Table 7](#), as the values for the extreme cases make the first experiment hard to compare with the others visually.

#### **(a) Cloudification phase**

The cloudification phase –graph (a)– performs better on EC2 than on the same MapReduce version in the local cluster for the three largest experiments (at least a 55% faster, in the worst case, up to a 74%). The smallest experiment, however, runs a 57% slower, mainly due to the execution time being too short to make up for the platform’s launching and synchronization overhead; this issue can be observed in all of the remaining stages.

#### **(b) Kernel execution**

The simulation execution stage, (b), is the most determinant phase in the whole process, ranging from the 46% of the whole execution time, in case I on EC2, to a 91%, in case III in the same environment. Since we selected our virtual cluster to match the number of cores of the physical environment, we can claim that the application deployed in the Cloud shows outstanding performance against the other configurations for the largest experiment (47% faster in the worst case against MRv2 on the local node). Experiments I, II and III, however, have similar execution times in EC2 and the physical node running MRv2, yet this sustains the scalability of the application, as no significant performance loss (14% tops) is perceived in average sized experiments after cloudification.

#### **(c) Aggregated time**

In (c) we observe the overall execution time for the application including both MapReduce jobs and input data upload. The latter has to be considered given that replication and balance must be achieved by the platform to distribute load evenly. The graph indicates that the performance obtained with MapReduce on Yarn in both the single-node cluster and the elastic cloud is remarkably better than the original multi-thread application –68% and 85% less total simulation time for the largest experiment, respectively–. The shared memory simulator’s results might be caused by the bottleneck constituted by the physical memory and the disk. The latter is particularly critical, as all threads write their results to disk while they perform their computations in the original simulator.

As we have already mentioned, the smallest experiment is an interesting exception, with execution times ten times greater than the original application in all the platforms. This reflects how the MapReduce framework’s overhead significantly affects the time taken to complete such a small simulation compared to the original application benchmark. (d) Scalability study

Finally, in (d) we observe the speed-up obtained on EC2 running YARN when the number of slaves is increased. The speed-up shown in the figure is related to the execution times commented in the previous paragraphs, which were obtained in a five-slave cluster. As the figure indicates, increasing the number of slaves decreases the total simulation time. However, the performance does not scale up linearly with the number of nodes: while with 16 nodes the speedup is 3.3, with 64 nodes it is only 7.6. The reason behind this result is that the problem size becomes small for the cluster size as more nodes are added. Hence, less data

is assigned to each slave and some resources become underutilised. Moreover, as we mentioned in the previous paragraph, in very small experiments the measured execution time is mostly spent in the platform's task preparation and scheduling, and not in the actual simulation. This results in degraded performance due to platform overhead. Therefore, it is necessary to increase the problem size as well as the number of slave nodes in order to achieve linear scalability.

## 6. Empowering the methodology: multidimensional analysis

Most simulators rely on several parameters to configure a specific experiment. For instance, the railway simulator presented in Section 4 requires an infrastructure definition to determine the number and position of power stations, the association between trains and tracks, the catenary sections and the number of trains in operation, among others. All of these variables can be changed independently, yielding an exponential number of possible experiments. This complexity is further problematic if one decides to combine more than one variable at a time in a multidimensional experiment analysis. Therefore, it is meaningful to consider multidimensional analysis within our methodology, thus providing a mechanism to execute concurrently a large number of simulations in the same infrastructure.

Since the methodology we described in Section 3 aims to provide task independence between concurrent simulation partitions, we can perform a multidimensional analysis by spawning several simulations in a many-task manner.

Many-task computing (MTC) is a new computing paradigm that mixes high throughput computing (HTC) and high performance computing (HPC). Its main goal is to make an efficient use of a large number of computing resources over short periods of time to execute many computational tasks [22]. The main difference with HTC is that the throughput is measured as tasks over very short periods of time –such as FLOPS, and tasks/s– instead of jobs per month, for instance.

The kinds of applications that benefit the most of this paradigm are the ones with many loosely coupled tasks, including heterogeneous tasks with some interdependencies. Most of these applications are focused on big data analysis on clusters, grids and supercomputers. For instance, scientific simulations such as DNA database analysis, data processing in the Large Hadron Collider (LHC) and climate modeling constitute examples of applications that make use of many-tasks to analyse their huge amount of data. These applications rely on large quantities of data, therefore data locality seems critical for large scale MTC. In fact, [22] states that data-aware scheduling minimizes data movement across nodes and benefits performance. Given this context, it seems natural to enhance our data-centric methodology with a many-task deployment to support multidimensional analysis, while enforcing efficient resource utilization and balance.

Finally, in previous works [34], the authors expose the advantages of developing simulators with an increased set of capabilities: not only simulating and testing scenarios provided by the user, but also generating a new set of scenarios by its own, thus exploring the solution search space. A simulator should also implement the evaluation function necessary to score the simulated

scenario, off-loading this task from the user. Combining these two properties, a third one arises, since with a little additional development the simulator should be capable of conducting a guided search for optimal solutions across the problem’s domain space.

Given this context, it seems natural to enhance our data-centric methodology with a many-task deployment to support multidimensional analysis, while enforcing efficient resource utilization and balance. The following sections describe, implement and evaluate a multidimensional analysis tool, implemented as a many-task deployment based on our cloudification methodology.

### 6.1. Description

In order to enhance our methodology implementing the many-task programming paradigm, we added some elements to the methodology structure proposed in Fig. 1. This enhancement is represented in Fig. 5. The idea behind the figure is to wrap the adaptation and simulation phases described in Section 3 with an scenario generator, an evaluator and a search engine that performs an iterative loop. In [34], a general structure which includes all these components is introduced, as well as a proposal to translate this structure into a cloud-based architecture. Therefore we use this architecture as a starting point for implementing our many-task enhancement.

There are two major goals to be accomplished. The first is to increase the number of scenarios simulated concurrently to support multidimensional analysis, instead of simulating just one case like in the aforementioned evaluation. By dispatching several concurrent simulations we follow a many-task paradigm and take further advantage of the Hadoop cluster resources. The second goal is to provide further functionality for the end user by permitting the exploration of the solution space generated by introducing variations of the initial scenario, and simulating each of them. Through the proposed approach we can iteratively evolve the initial scenario in order to find better solutions according to an user-defined evaluation function and a search engine.

The elements included in the deployment’s architecture are the following:

Adaptation and simulation stages. These phases correspond to the ones belonging to the methodology described in Section 3, with a few considerations.

First, we could make use of different adaptation procedures, each tailored for a subset of the generated experiments datasets. However, the simulation kernel would remain the same for every experiment, allowing to execute the same simulator while varying several parameters at the same time.

Second, we remark every task is independent between experiments. This means we can execute the adaptation stage of one experiment at the same time we run the simulation stage of a different one. Therefore, we can interleave many heterogeneous tasks as in MTC paradigms.

This approach allows us to perform a better usage of the cluster resources, since a double-grained workload is distributed

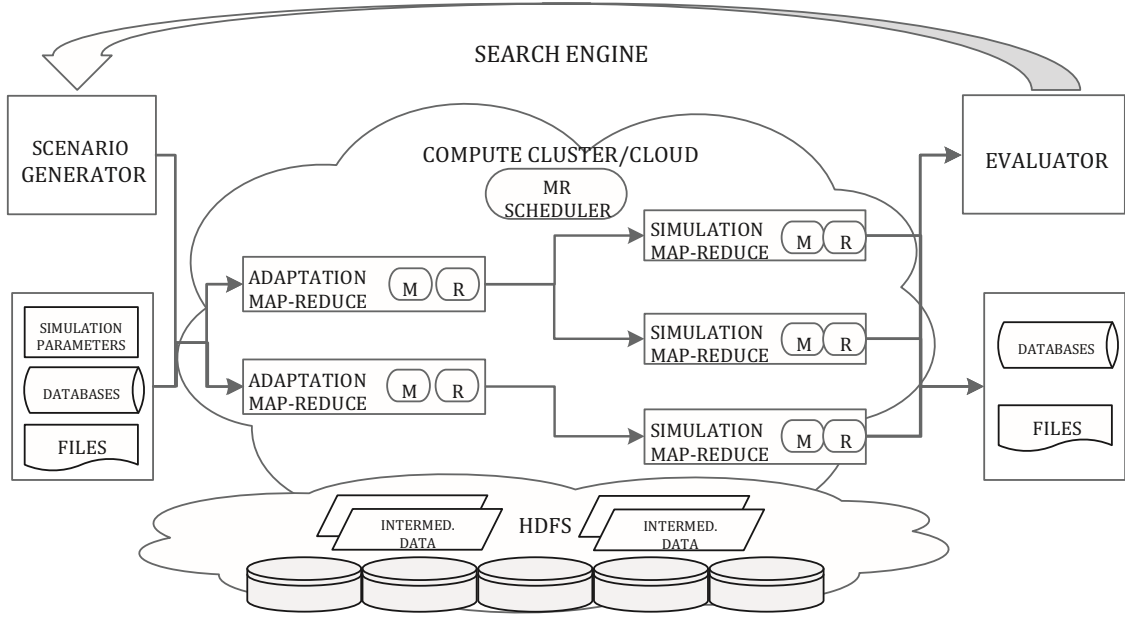


Fig. 5. Methodology enhancement including many-task deployment and optimum search for multidimensional analysis.

across the nodes. First, we run multiple simulation kernels (maps) which correspond to the same simulation scenario (i.e. the same map-reduce task). Second, we run multiple map-reduce jobs, since we are simulating several scenarios concurrently. Note that, since multiple map-reduce jobs can be on execution at the same time, the Hadoop task scheduler has a potential impact on performance across the global workload.

Scenario generator and evaluator. We propose that an efficient simulator should evaluate and simulate a set of solutions with a minimal user involvement. New generation simulators should be capable of proposing and evaluating new designs based on a range of possible parameters. Besides, generating new scenarios to be tested allows us to deploy the many-task paradigm in an easy way, with minimal user involvement. The proposed methodology aims these objectives through introducing two new components: a scenario generator and an evaluator. We define scenarios as independent simulations, each one of them with a different input data set (though parts, or even almost of the data can be similar), that have to be evaluated separately. Varying the input data leads to a different scenario (e.g. a different infrastructure to be tested, or different environmental conditions the current infrastructure have to be checked with).

The scenario generator and the evaluator wrap the simulation model (i.e. the adaptation and simulation phases) generating different solutions to be evaluated. The scenario generator creates new scenarios through variations in the input data, thus allowing experimentation with different simulation parameters, components or domain restrictions. Those scenarios are provided to the map-reduce cluster, which performs the simulation as described in Section 3. The evaluator analyses the output from that simulations and scores the generated solutions, stating whether they are acceptable or providing a measure of their quality.



Generating and evaluating multiple scenarios automatically allows a simulator to test different solutions, thus providing a faster way of exploring the solution space. Rather than obtaining a single solution, this method obtains a set of feasible solutions from which the user can select the best one. Moreover, advanced search algorithms may be implemented. For instance, the generator and the evaluator could implement a guided search using heuristics in order to find an optimal solution.

## 6.2. Case study

The cloudified application selected to perform the multidimensional analysis was the railway electric power consumption simulator presented in Section 4. The implementation proceeded as follows: starting from an initial test case, the scenario generator introduced variations in some key parameters of the railway infrastructure simulation. Specifically, the position of the power supply stations that feed the railway infrastructure was modified, thus generating a series of child test cases to be scored by the evaluator. Hence, the simulation dimensions are time, number of power stations and their position.

The simulation result which we aimed to optimize was the mean useful voltage, described in European normative UNEEN-50388 [35]. This measure is defined as the mean of all voltages at the pantograph of each train in the geographic zone, at each simulation time step. Let  $t = ft_1; t_2; \dots; t_{pg}$  be the simulation steps, and let  $n_i = fn_{i1}; t_{i2}; \dots; t_{iqg}$  be the number of trains in the scenario operating at step  $t_i$ . Thus, the mean useful voltage  $U_{mu}$  is calculated following Eq. 4, where  $U_{ij}$  is the voltage at

Table 8. CENELEC test case definition.

Trains	Tracks	Electrical substations	Circuit branches (mean)	Simulated time	Input size (MB)
6	2	3	150	1 h 20 min.	4.2

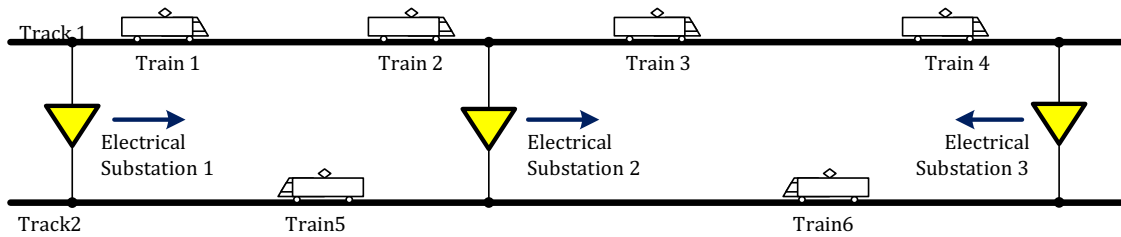


Fig. 6. Schema of the main railway elements in the CENELEC test case. Parallel connections between catenaries or tracks are not shown.

pantograph calculated at step  $t_i$  for train  $t_{ij}$ . This measure indicates the quality of the power supply. The lower the mean useful voltage is, the less energy is transferred from the supply stations to the

trains, on average. The many-task workload consisted of finding a solution which maximizes this value.

In order to find that solution, we conducted a breath-first search. Starting from the an initial test case, the position of the first electrical substation is modified by displacing its position 1 km., then 2 km, etc. Once all possible positions of this first electrical substation have been tested, we start to modify the position of the second electrical substation as well. We established a maximum number of generated scenarios, in order to reduce the solution search space. Improved search algorithms may be implemented in order to reach an optimum more efficiently. In this case, implementing an easy algorithm is more appropriate since now we are evaluating the scalability and performance of the solution.

The initial test case from which we modify the power stations position is none of the described in [Table 1](#). Instead, we use a standard railway scenario described in the proposed draft of the European normative UNE-EN-50641.<sup>1</sup> This proposal of normative establishes the requirements for the validation of simulation tools used for the design of traction power supply systems. Therefore, it is meaningful to apply such normative to our cloudified railway simulator. Composing elements of this case are described in [Table 8](#), and an schema describing the railway elements of the test case is shown in [Fig. 6](#). Notice that not all the branches of the electric circuit are shown in this schema. The arrows in the diagram indicate the displacement direction of the electrical substations as more cases are generated and evaluated.

## 7. Multidimensional analysis evaluation

We performed a second battery of tests in order to illustrate the many-task deployment capabilities of our methodology. In the first evaluation we only executed one of the test cases at a time. This was conducted to study the behavior of the workload distribution in a cloud and cluster. In this evaluation, many experiments will be spawned in the same Hadoop cluster, following the MTC paradigm. While the first evaluation focused on the speed-up and computing time required to perform a single test case, this second evaluation was centered in the number of test cases simulated simultaneously and the obtained scalability and throughput.

The selected platform to deploy the evaluation of the multidimensional analysis was MRv2 on Amazon EC2. MRv2 outperformed MRv1 in the first evaluation, getting better results than its counterpart in all tests. Amazon EC2 was selected in order to take advantage of the cloud's possibility to allocate resources (slave nodes in this case) on demand. The virtual cluster consisted of a general purpose m3.xlarge node as dedicated master and memory optimized m2.xlarge machines (see [Section 5](#)) as slaves. These tests have been conducted using four virtual clusters of 1, 4, 16, and 64 slaves respectively. In each virtual cluster we perform tests simulating 1, 4, 16, and 64 experiments, always following the breadth-first search pattern aforementioned.

As shown by [Fig. 7](#), our tests indicate that the enhanced system scales linearly with the number of experiments for every cluster size tested. This suggests that the proposed methodology is

---

<sup>1</sup> This normative is still a proposal under vote by the CENELEC committee until 30th of January, 2015, when it will be finally either approved or discarded.

suitable for multidimensional analysis via MTC, as we can run several interleaved heterogeneous tasks making an efficient use of the infrastructure's resources. Furthermore, we can see that linearity is not loss in any case, so the end user can run the many-task job successfully with either a few nodes or a

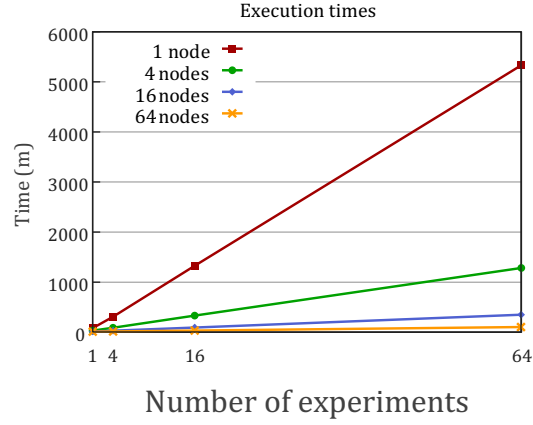


Fig. 7. Execution times for the enhanced methodology with increasing number of nodes and experiments.

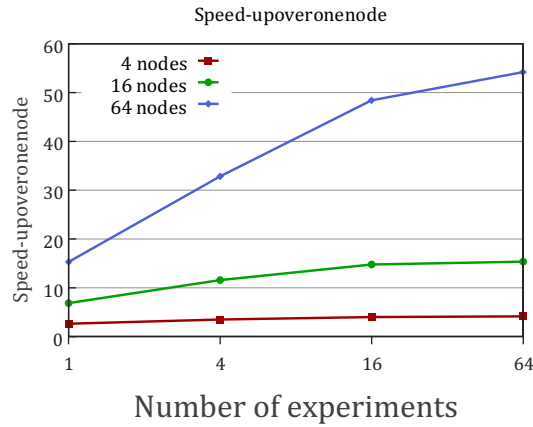


Fig. 8. Speed-up for the enhanced methodology, over one node.

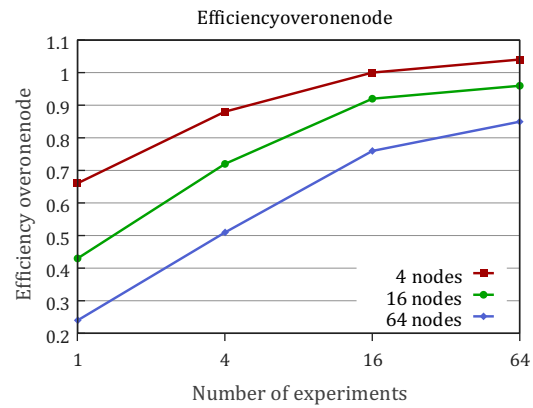


Fig. 9. Per node efficiency of the enhanced methodology with increasing number of experiments.

larger infrastructure. That allows to tailor the underlying infrastructure to reduce of economical costs or provide higher performance.

In Fig. 8 we show the resulting speed-up of the enhanced deployment, taking as base a single-node execution with increasing number of nodes and experiments. In Fig. 9 we include the per node efficiency,  $e$ , which corresponds to the normalized speed-up with relation to the number of slaves. The worst result corresponds to 64 nodes and one experiment because of the significant impact of the Hadoop platform's overhead in the overall execution time. Noticeably, with four nodes and 64 experiments we obtain superlinear speed-up due to the efficient utilization of the cluster resources. As seen in Fig. 9, when more nodes are added, the system becomes underused and the platform's overhead becomes noticeable, thus losing efficiency. The same situation occurs if we run less experiments in the same environment. Therefore, the overall system's efficiency is tightly related to the ratio between the number of nodes and experiments. It is also remarkable that the system increases its efficiency as more experiments are executed, and that it is not needed to rely on a large cluster to execute them if efficiency is more relevant than total execution time.

## 8. Conclusions

As the cloud is increasingly shown as a viable alternative to traditional computing paradigms for high-performance applications and resource-intensive simulations, we propose a general methodology to transform numeric simulations into a highly scalable MapReduce application that re-uses the same simulation kernel while distributing the simulation load across as many nodes are desired in a virtual cluster running on the cloud. The procedure requires an application analysis phase in which at least one independent variable must be found, since this element will act as index for the cloudification phase. The cloud adaptation stage transforms the original input into a set of partitions indexed by the previous variable by means of a MapReduce job; these partitions are fed to a second MapReduce job that executes the simulation kernel independently for each, merging the final results as well.

This methodology performs a paradigm shift from resource-bound applications to a data-centric model; such cloudification mechanism provides effective cloud migration of simulation kernels with minimal impact on the original code and achieves great scalability since limiting factors are scattered. Therefore, it provides a way to increase application's sustainability, breaking the dependence on local infrastructure, and allowing to spread simulation scenarios of different sizes in a more flexible way, using heterogeneous hardware, and taking advantage of shared inter-domain infrastructures.

Additionally, we provide a way to enhance the methodology to support multidimensional analysis. This allows the generation, scheduling, execution and evaluation of many different experiments for a single simulator. We implemented this new scheme following a many-task model. The results we obtained show that the more parallel executions we have, the more efficient is the infrastructure's resources utilization.

Future works are strongly focused on extending the current methodology to a generalized framework which would allow to cloudify any scientific application. With this aim, several issues have to be solved.

First of all, Hadoop MapReduce is constrained to a specific set of operations that limit the flexibility of the methodology. This is one of the reasons why more advanced and complex platforms are arising nowadays to substitute Hadoop and MapReduce [37]. We are considering other implementations to migrate to, such as Spark, that provide further functionality and support for complex algorithms that require iterative MapReduce procedures.

Secondly, the behavior of the methodology should be analysed with other kind of applications (CPU or network intensive). Currently we are cloudifying a classic MPI application, the n-bodies problem, in order to assure performance even in clusteroriented applications. Moreover, parameter extraction and application analysis is currently performed manually by the user, who is accountable for selecting an independent variable  $T_x$ . Current development is also oriented to ease this tasks through creating data definitions which would allow the adaptation phase to select and split the input data automatically.

## Acknowledgements

This work has been partially funded under the grant TIN2013-41350-P of the Spanish Ministry of Economics and Competitiveness, and the COST Action IC1305 “Network for Sustainable Ultrascale Computing Platforms” (NESUS).

## References

- [1] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, et al., The opportunities and challenges of exascale computing, in: Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, November 2010.
- [2] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, et al., Exascale computing study: technology challenges in achieving exascale systems, Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15.
- [3] K. Yelick, S. Coghlan, B. Draney, R.S. Canon, et al., The Magellan Report on Cloud Computing for Science, US Department of Energy, Washington DC, USA, Tech. Rep.
- [4] P. Mell, T. Grance, The nist definition of cloud computing, *Natl. Inst. Stand. Technol.* 53 (6) (2009) 50.
- [5] N. Grozev, R. Buyya, Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey, *Software: Practice and Experience*.
- [6] D. Petcu, G. Macariu, S. Panica, C. Craiciun, Portable cloud applications from theory to practice, *Future Gener. Comput. Syst.* 29 (6) (2013) 1417–1430.
- [7] S. a. Caíno, A cloudification methodology for numerical simulations, in: Proceedings of the 2014 1st Workshop on Techniques and Applications for Sustainable Ultrascale Computing Systems (TASUS), 2014.
- [8] S. Caino, A. Garcia, F. Garcia-Carballera, J. Carretero, Breaking data dependencies in numerical simulations using mapreduce, in: XXV Jornadas de Paralelismo, 2014.
- [9] J. Ekanayake, S. Pallickara, G. Fox, Mapreduce for data intensive scientific analyses, in: IEEE Fourth International Conference on eScience, 2008, eScience '08, 2008, pp. 277–284. <http://dx.doi.org/10.1109/eScience.2008.59>.

- [10] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins, Pig latin: a not-so-foreign language for data processing, in: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08*, ACM, New York, NY, USA, 2008, pp. 1099–1110, <http://dx.doi.org/10.1145/1376616.1376726>.
- [11] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, R. Murthy, Hive – a petabyte scale data warehouse using hadoop, in: *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, 2010, pp. 996–1005. <http://dx.doi.org/10.1109/ICDE.2010.5447738>.
- [12] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, Bigtable: a distributed storage system for structured data, *ACM Trans. Comput. Syst.* 26 (2) (2008) 4:1–4:26, <http://dx.doi.org/10.1145/1365815.1365816>.
- [13] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10*, Berkeley, CA, USA, 2010, pp. 10–10.
- [14] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, G. Fox, Twister: A runtime for iterative mapreduce, in: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10*, ACM, New York, NY, USA, 2010, pp. 810–818, <http://dx.doi.org/10.1145/1851476.1851593>.
- [15] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, P. Maechling, Scientific workflow applications on amazon ec2, in: *2009 5th IEEE International Conference on E-Science Workshops*, 2009, pp. 59–66. <http://dx.doi.org/10.1109/ESCIW.2009.5408002>.
- [16] Z. Hill, M. Humphrey, A quantitative analysis of high performance computing with amazon's ec2 infrastructure: The death of the local cluster? in: *2009 10th IEEE/ACM International Conference on Grid Computing*, 2009, pp. 26–33. <http://dx.doi.org/10.1109/GRID.2009.5353067>.
- [17] T. Gunarathne, T.-L. Wu, J. Qiu, G. Fox, Mapreduce in the clouds for science, in: *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 565–572. <http://dx.doi.org/10.1109/CloudCom.2010.107>.
- [18] S.N. Srirama, P. Jakovits, E. Vainikko, Adapting scientific computing problems to clouds using mapreduce, *Future Gener. Comput. Syst.* 28 (1) (2012) 184–192, <http://dx.doi.org/10.1016/j.future.2011.05.025>.
- [19] S. Seo, E. Yoon, J. Kim, S. Jin, J.-S. Kim, S. Maeng, Hama: an efficient matrix computation with the mapreduce framework, in: *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 721–726. <http://dx.doi.org/10.1109/CloudCom.2010.17>.
- [20] P. Xuan, Y. Zheng, S. Sarupria, A. Apon, Sciflow: a dataflow-driven model architecture for scientific computing using hadoop, in: *2013 IEEE International Conference on Big Data*, 2013, pp. 36–44. <http://dx.doi.org/10.1109/BigData.2013.6691725>.
- [21] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, Heuristics for scheduling parameter sweep applications in grid environments, in: *Heterogeneous Computing Workshop, 2000, (HCW 2000) Proceedings*, 9th, IEEE, 2000, pp. 349–363.
- [22] I. Raicu, I. Foster, Y. Zhao, Many-task computing for grids and supercomputers, in: *Workshop on Many-Task Computing on Grids and Supercomputers*, 2008, MTAGS 2008, 2008, pp. 1–11. <http://dx.doi.org/10.1109/MTAGS.2008.4777912>.
- [23] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, Performance analysis of cloud computing services for many-tasks scientific computing, *IEEE Trans. Parallel Distrib. Syst.* 22 (6) (2011) 931–945, <http://dx.doi.org/10.1109/TPDS.2011.66>.
- [24] D. de Oliveira, E. Ogasawara, F. Baiao, M. Mattoso, Scicumulus: a lightweight cloud middleware to explore many task computing paradigm in scientific workflows, in: *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, 2010, pp. 378–385. <http://dx.doi.org/10.1109/CLOUD.2010.64>.
- [25] G. D'Angelo, Parallel and distributed simulation from many cores to the public cloud, in: *2011 International Conference on High Performance Computing and Simulation (HPCS)*, 2011, pp. 14–23. <http://dx.doi.org/10.1109/HPCSim.2011.5999802>.

- [26] D. Yu, J. Wang, B. Hu, J. Liu, X. Zhang, K. He, L.-J. Zhang, A practical architecture of cloudification of legacy applications, in: 2011 IEEE World Congress on Services (SERVICES), 2011, pp. 17–24. <http://dx.doi.org/10.1109/SERVICES.2011.84>.
- [27] S. Srirama, V. Ivanistsev, P. Jakovits, C. Willmore, Direct migration of scientific computing experiments to the cloud, in: 2013 International Conference on High Performance Computing and Simulation (HPCS), 2013, pp. 27–34. <http://dx.doi.org/10.1109/HPCSim.2013.6641389>.
- [28] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, *Commun. ACM* 51 (1) (2008) 107–113.
- [29] J. Carretero, J.M. Pérez, F. García-Carballeira, A. Calderón, J. Fernández, J.D. García, A. Lozano, L. Cardona, N. Cotaina, P. Prete, Applying RCM in large scale systems: a case study with railway networks, *Reliab. Eng. Syst. Safety* 82 (3) (2003) 257–273.
- [30] T. White, *Hadoop: The Definitive Guide: The Definitive Guide*, O'Reilly Media, 2009.
- [31] K. Kambatla, A. Pathak, H. Pucha, Towards optimizing hadoop provisioning in the cloud, in: Proc. of the First Workshop on Hot Topics in Cloud Computing, 2009, p. 118.
- [32] K. Yamazaki, R. Kawashima, S. Saito, H. Matsuo, Implementation and evaluation of the jobtracker initiative task scheduling on hadoop, in: 2013 First International Symposium on Computing and Networking (CANDAR), 2013, pp. 622–626. <http://dx.doi.org/10.1109/CANDAR.2013.112>.
- [33] T.A.S. Foundation, Apache hadoop 2.2.0, October 2013. <<http://hadoop.apache.org/docs/stable/>>.
- [34] A. García, C. Gómez, F. García-Carballeira, J. Carretero, Enhancing the structure of railway infrastructure simulators, in: Proceedings of the 1st International Conference on Engineering and Applied Sciences Optimization (OPT-i), 2014, pp. 352–363.
- [35] BS-EN-50388, Railway Applications, Power Supply and Rolling Stock, Technical Criteria for the Coordination Between Power Supply (Substation) and Rolling Stock to Achieve Interoperability., Tech. rep., 2012.
- [36] T. Gunarathne, T.-L. Wu, J.Y. Choi, S.-H. Bae, J. Qiu, Cloud computing paradigms for pleasingly parallel biomedical applications, *Concurrency Comput.: Pract. Experience* 23 (17) (2011) 2338–2354.
- [37] C. Chambers, A. Raniwala, F. Perry, S. Adams, R.R. Henry, R. Bradshaw, N. Weizenbaum, Flumejava: easy, efficient data-parallel pipelines, *ACM Sigplan Notices*, vol. 45, ACM, 2010, pp. 363–375.