Caíno-Lores, S., Lapin, A., Carretero, J., Kropf, P. (2020). Applying big data paradigms to a large scale scientific workflow: Lessons learned and future directions. *Future Generation Computer Systems*, 110, pp. 440-452.

DOI: 10.1016/j.future.2018.04.014

Applying Big Data Paradigms to a Large Scale Scientific Workflow: Lessons Learned and Future Directions

S. Caíno-Lores, J. Carretero
Department of Computer Science, University Carlos III of Madrid
A. Lapin, P. Kropf
Computer Science department (IIUN), University of Neuchâtel

Abstract

The increasing amounts of data related to the execution of scientific workflows has raised awareness of their shift towards parallel data-intensive problems. In this paper, we deliver our experience combining the traditional high-performance computing and grid-based approaches with Big Data analytics paradigms, in the context of scientific ensemble workflows. Our goal was to assess and discuss the suitability of such data-oriented mechanisms for production-ready workflows, especially in terms of scalability. We focused on two key elements in the Big Data ecosystem: the data-centric programming model, and the underlying infrastructure that integrates storage and computation in each node. We experimented with a representative MPI-based iterative workflow from the hydrology domain, EnKF-HGS, which we re-implemented using the Spark data analysis framework. We conducted experiments on a local cluster, a private cloud running OpenNebula, and the Amazon Elastic Compute Cloud (AmazonEC2). The results we obtained were analysed to synthesize the lessons we learned from this experience, while discussing promising directions for further research.

1 Introduction

Scientific workflows are key tools in many research areas that rely on multiple, diverse, and distributed operations over various datasets, usually yielding major computational complexity and data dependencies. Nowadays, the increasing amount of input data, intermediate data, and even output data related to the execution of workflows is shifting these originally computationally intensive systems towards parallel data-intensive problems.

While current workflows rely on hundreds of gigabytes of intermediate data [45], trends show that large scale workflows would have to address increasing data sizes, easily reaching petascale [56]. In this context, scientific workflows face new performance and scalability challenges in terms of data management, workload distribution, load balance, and scheduling [35].

Given the data-intensive nature of these problems, recent works have suggested the opportunity of combining the traditional high-performance computing (HPC) and grid-based approaches with Big Data (BD) analytics and high-throughput (HTC) paradigms [28]. For example, typical BD programming models –such as Apache Hadoop[1]– have been considered to substitute MPI parallelism induction mechanisms, following a data-centric approach. In addition, we can also see this opportunity affecting the underlying computing infrastructures. Indeed, cloud computing (CC) –a key element in current data analytics systems– could inspire hybrid platforms for exascale scientific workflows in which storage is not completely isolated from computing nodes [40].

As an example, there has been a rise of scientific many-task computing (MTC) workflows [16], which are capable to handle the huge data volume and the massive computational requirements of these simulations. A Survey of Data-Intensive Scientific Workflow Management -> can be cited pretty much anywhere where we need to support the viability of cloud or bd techniques for data intensive workflows, also contains sevral workf management examples [28] Following this trend, BD infrastructures and paradigms are increasingly seen as alternatives to traditional HPC approaches for some major types of scientific applications, especially those with many loosely-coupled tasks [36], or heterogeneous tasks with few interdependences [39]. In previous works [8]

[1] Apache Hadoop is available at http://hadoop.apache.org/

we showed that applying some of these mechanisms could improve scalability in parameter-based scientific simulations. In particular, we focused on increasing the addressable size and complexity of standalone scientific applications, executed within map-reduce-based wrappers.

The scientific community is currently interested in studying the potential of these BD techniques to scale-up scientific workflows. In this work we focus on the family of *scientific ensembles* [13], which includes many-task workflows like CyberShake [9], Montage [14] and Materials Project [23]). However, the architectural differences between the analytics and scientific worlds might require novel approaches to achieve satisfactory results. Therefore, in this work we assess the suitability of such data-oriented mechanisms for production-ready workflows, especially in terms of scalability.

To study this, we reproduced the functionality of an iterative scientific simulator from the hydrology domain implemented in MPI, EnKF-HGS [26], in Apache Spark 1.6.0[2], which is currently a major representative of the data-centric analytics ecosystem [52], and similar in performance to other platforms [32]. This work is an extension of a previous preliminary assessment in which we analysed the behaviour of the original and the Spark implementations of the former simulator in a traditional cluster [10]. Here we add further evaluations against two types of cloud infrastructures: a private cloud running OpenNebula, and the AmazonEC2 public cloud. We evaluated scalability, and analysed the behaviour of the platform and the infrastructure as the problem size increased.

The goal of this paper is to present and discuss our experience with the application of these paradigms, platforms and infrastructures currently used in BD, to a typical scientific HPC workflow. As a result, we provide the following contributions:

- An evaluation of a typical simulation ensemble at two levels: execution model –Spark against MPI– and computing model –cluster against public and private cloud–.
- An analysis of the strengths, weaknesses and limitations of the Big Data and HPC paradigms derived from the former results and the state of the literature.
- An identification of the most relevant future directions in the context of current scientific interests and challenges.

The rest of this paper contains background on the BD techniques we relied on and their relation with scientific workflows (Section 2); A high-level description of the workflow from the hydrology domain we considered as use case (Section 3); Key architectural details of the simulation ensemble implemented in Spark (Section 4); and the evaluations we conducted to study the behaviour of the resulting redesigned workflow (Section 5). Finally, this paper concludes with a discussion of the lessons learned from this experience (Section 6) and future directions (Section **??**), and a summary of this work (Section 7).

2 Background and Related Work

In this Section we comment some relevant references related to our work covering four aspects: scientific workflows, cloud computing, data analytics platforms, and experiences with HPC and Big Data.

2.1 Scientific Workflows

The concept of scientific workflow emerged in response to automating large-scale computational experiments in different domains. A scientific workflow aims to organize computational steps into a logical series in order to prove a scientific hypothesis. A good representative of a system, which provides an execution environment for a scientific workflow and tools for data management, analysis, simulation, and visualization, is a simulator. Simulators firstly emerged in meteorology and nuclear physics, then they became crucial in many other disciplines, e.g., economics, sociology, biology, geology, hydrology [51]. Simulators differ based on four main types of simulation problems defined in the literature [4, 38, 44]. While equation-based and agent-based simulations are widely adopted and well-studied types, two other types – multiscale and Monte Carlo simulations – were developed later due to the requirement of much greater computational power.

[2]Apache Spark 1.6.0 documentation is available at http://spark.apache.org/docs/1.6.0/

2.2 Cloud Computing

Cloud computing (CC) is a popular paradigm that relies on resource sharing and virtualization to provide the end-user with a transparent, scalable and elastic system that can be expanded or reduced on-the-fly. It emerged with the idea of virtually unlimited resources obtainable on-demand [34], and it has been widely adopted in the BD community as a solution for both heavy and variable analytics workloads. This popularity is a consequence of some of the core features of cloud service models, such as:

- Minimal management effort, as the infrastructure is maintained and administrated by a third-party.
- Automatic or manual scale up or down according to utilization, thus supporting elasticity.
- Potential to reduce economical costs, as it follows a pay-as-you-go model.

Several works have addressed the opportunities of shifting scientific workflows from traditional HPC and HTC infrastructures to CC platforms. In particular, authors have focused on exploring data-intensive workflows, since they are the most tightly related to conventional BD applications in terms of data volumes [55, 27]. Experimentation with well known workflows, like Montage, shows that running costs could be significantly decreased with CC infrastructures, but performance would suffer from virtualization and latency overheads [15, 22, 3].

Large-scale scientific applications often need to process enormous amount of data in the terabyte or even petabyte range and require special high performance hardware with low latency connections to complete computation in a reasonable amount of time. To address these challenges, we build an infrastructure that can dynamically select high performance computing hardware across institutions and dynamically adapt the computation to the selected resources to achieve high performance

2.3 Data Analytics Platforms

Scientific workflows are composed of heterogeneous and coupled components that simulate different aspects of the domain they model. These modules interact and exchange significant volumes of data at runtime, hence making these transfers efficient has a potential major impact in the overall performance of the resulting application [54]. As a consequence, both the storage infrastructure and the logical file system abstractions could affect performance and scalability, thus making data management a key aspect in workflow design and implementation [47].

Big Data analytics tools –like Hadoop and Spark– are being explored to provide straightforward data distribution and caching mechanisms in data-intensive HPC applications. Their data-centric nature permits reasoning about tasks over distributed datasets without worrying about task scheduling, which is managed by the middleware to enforce data locality and minimize transfers. The inherent parallelism of these tools has resulted in positive experimental results showing their suitability for massively parallel workflows [53]. Nevertheless, challenges remain with respect to workflows built with a pure HPC focus, which rely on MPI and traditional storage infrastructures [30].

These tools have already been used to create workflow execution engines and scientific workflow management systems (SWfMS) for current state-of-the-art workflows [7, 53]. However, this topic is still fairly new and the experience with applying these techniques is still limited. Previous works have contributed with guidelines and methodological approaches to make the design of scientific workflows easier and more efficient, with a user-centric and visual perspective [17]. A theoretical analysis of migrating common HPC-oriented workflows to a BD processing platform (i.e., Apache Hadoop) was made in [13]. Authors implemented six representatives of common scientific workflow patterns in Apache Hadoop environment and discussed implementation challenges as well as Hadoop environment applicability for each of the basic patterns.

2.4 HPC and Big Data Paradigms

Several previous works have been devoted to the application of BD frameworks, specially Map Reduce, to the HPC world. Most attempts have tried to adapt map-reduce to use distributed file systems available in HPC environments. For instance, the Ceph file system provides support for Hadoop, and Maltzahn et al. [31] studied the combination of both systems. The result of running

map-reduce with HDFS and GPFS was studied by Ananthanarayanan et al. [1] and Wang et al. [49]. These works investigated the data-centric analytics framework running on a compute-centric HPC environment that relies on high-performance file systems.

The performance of executing HPC applications on data-centric cloud has also been studied in several works. The results of executing a scientific HPC application on Amazon EC2 were presented by Evangelinos et al. [18], showing that the performance of network in cloud is worse than that of HPC by one to two orders of magnitude, which was also proved by Gupta et al. [21] on different cloud platforms. Both showed that raw performance difference between compute centric HPC and cloud is pronounced.

In this work, we focus not only in the design and deployment of Big-Data inspired scientific workflows, but also in the performance and scalability issues they inherit from the platform and infrastructure. Besides that, we aim to provide insight on the potential benefits of redesigning HPC-oriented workflows to BD platforms by indicating potential issues and bottlenecks, which might arise from a direct execution of an HPC application in cloud environment. Thus, our work is different than the previously cited contributions, as they had mostly studied usage of existing platforms for pleasingly parallel problems in the case of data or the execution of HPC applications in the cloud.

3 The EnKF-HGS Use Case

Studies have shown the feasibility of running cloud-based frameworks for multi-scale data analysis while preserving the performance and storage capability of grids [29, 50]. The hydrology domain is an example of the former that has already been assessed in cloud environments [12]. A wide range of hydrological problems has been proved suitable for their execution in hybrid computing infrastructures integrating grids with external cloud providers. This covers both computationally intensive HPC simulators, and MTC-like applications with multiple scenarios. Nevertheless, to the best of our knowledge, data analysis techniques have not been applied yet to this domain, which we consider complex enough to represent the family of scientific workflows that rely not only on large volumes of data, but also on major computational resources like CPU and memory.

As mentioned in the Subsection 2.1, simulators can be considered as a good representative of scientific workflows. Taking into account four simulation types, we selected the one, which covers different scientific problems [**?**, **?**, **?**, **?**]. All these problems have one particular similarity – they can be solved by the Monte Carlo method. Even though the simulators directed at solving these problems vary but they all share the same workflow structure, which involves a large random sampling to determine the properties of some phenomenon or system behaviour. An example of such a simulator can be the EnKF-HGS simulator [25].

3.1 Simulator Description

EnKF-HGS provides functionality for real-time stochastic simulations of the groundwater and surface water profiles, with an optional real-time control of water resource systems through a feedback mechanism. The core of the simulator is the data assimilation process, which allows to incorporate observations of an actual environmental system into a numerical model of that system. This process continuously improves the simulated model and minimizes the deviation of the model from the state of the actual system. In the EnKF-HGS simulator, data assimilation is implemented via the ensemble Kalman Filter [19, 6], which approximates the uncertainty of model prediction through the forward simulation of an ensemble of model realizations. Each model realization represents an instantiation of the numerical model of the environment with a different combination of input parameters and system state conditions. In our case the numerical model is the integrated hydrological modelling software HydroGeoSphere (HGS) [46]. HGS is able to perform dynamic stochastic simulations between water profiles composed of numerous elements, as depicted in Fig. 1. It has been successfully used to simulate many complex problems involving surface water, groundwater and vegetation processes [37, 42]. The model is designed to take into account all key components of the hydrologic cycle using a rigorous conceptualization of the hydrologic system [5, 33]. Based on these parameters, HGS allows the simulation of a wide range of physical characteristics and hydrological objects, such as wells, tile drains, and thermal energy transport.

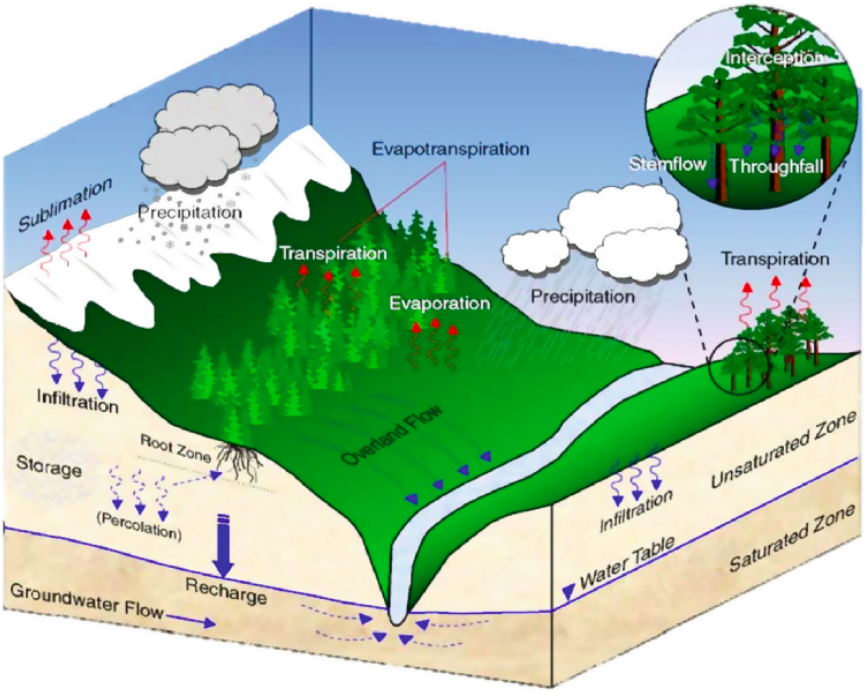


Figure 1: Typical surface water and groundwater processes in a pre-alpine type of valleys [24].

The overall EnKF-HGS workflow and the original simulator implementation details are presented in Subsection 3.2.

Considering the latest generation of hydrological numerical models, one of the most advances codes in this respect is HydrogeoSphere HGS is able to perform dynamic stochastic simulations between water profiles composed of numerous elements, as depicted in Fig. 1. It has been successfully used to simulate many complex problems involving surface water, groundwater and vegetation processes [37, 42].

The simulator heavily relies on HydroGeoSphere (HGS), which is a numerically demanding code implementing a 3D control-volume finite element hydrology model with a fully integrated surface-subsurface water flow and solute, including thermal energy transport. It provides functionality for dynamic stochastic simulations between the water profiles composed of numerous elements, as depicted in Figure 1. HGS has been successfully used to simulate many complex problems involving surface water, groundwater and vegetation processes [37, 42].

HGS uses the control volume finite element method to solve the flow equations for all domains considered in a simulation. Non-Linear equations are solved by means of the Newton-Raphson linearisation method, and the matrix equation arising from the discretization is solved by a preconditioned iterative solver. As shown in Eq. 1, the execution control is conducted with a variable time-stepping procedure. This is defined in HGS according to the rate of change of the solution variable (e.g., flow, temperature), so that the simulation uses increasingly larger time steps to reduce the number of steps towards convergence, and thus the computing time, if the dependent variable, V, does not experience drastic changes in several iterations.

$$\Delta_t^{k+1} = \frac{V_{max}}{max|V_i^{k+1} - V_i^k|}\Delta_t^k \tag{1}$$

Figure 2 depicts main operations of the EnKF-HGS workflow. EnKF-HGS allows to constantly improve the simulated model by sequentially assimilating (i.e. Filtering phase of EnKF) fresh field measurements (i.e. Read observations) and re-executing HGS (i.e. Simulate realisations) with updated data. Specifically, a sequential update of system states and model parameters guarantees a continuous improvement of the model predictions.

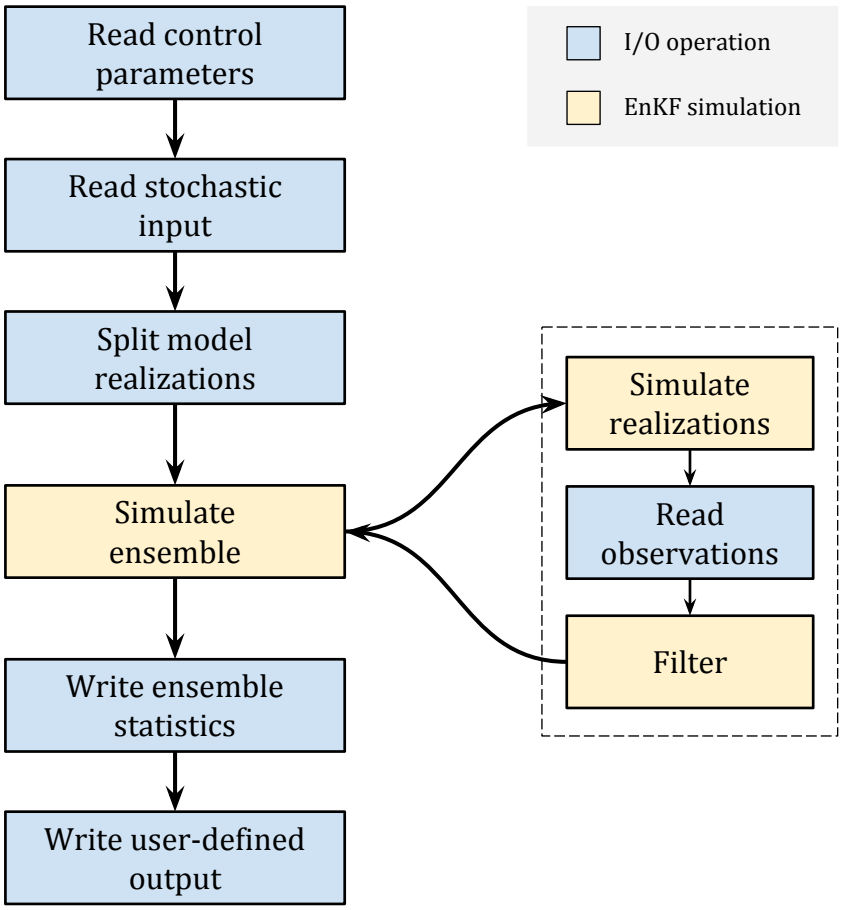


Figure 2: Main operations of the EnKF-HGS workflow.

3.2 Original Simulator Workflow and Implementation

The ensemble Kalman filter is an implementation of the Monte Carlo method that consist of two distinct steps: (i) the forward propagation of the ensemble of model realisations (i.e. forward propagation phase), and (ii) an update of the simulated model state with the measurements (i.e. filtering phase). The filtering phase is a relatively short-lasting but tightly-coupled process that performs a set of matrix operations and requires multiple data synchronization points. While the forward propagation phase comprises a large pool of independent model realizations, which introduces a tremendous demand for computing power, especially if combined with a complex numerical model such as HydroGeoSphere. On top of that, the iterative nature of the data assimilation process imposes that the two phases have to be repeated continuously, thus shifting the demand even further. Even though this method results in higher quality model predictions than the conventional simulation methods, the high resource demand of the method remains an unsolved problem for many environmental scientists. This makes EnKF-HGS simulator a perfect representative of a complex and compute-intensive scientific application. EnKF-HGS performs the two aforementioned steps of the ensemble Kalman filter with a couple of implementation specific auxiliary steps shown in Fig. 2. Without a carefully designed parallelization strategy, obtaining any simulation results within a reasonable execution time would be absolutely infeasible. Due to that the original simulator implementation uses the MPI framework for distributing the simulations of the ensemble members over available CPUs as each individual member represents a completely self-contained instantiation of the model. To be more precise, one simulation of the HydroGeoSphere model comprises the sequential execution of two proprietary simulation kernels: GROK and HGS [46, 5]. Where GROK is a preprocessor that prepares the input files for HGS, which makes GROK an I/O intensive application. HGS, on the other hand, is an integrated hydrological modelling simulator, which mainly relies on the CPU to solve the aforementioned differential equations.

Figure 3 shows a simplified execution model of the MPI implementation of the EnKF-HGS.

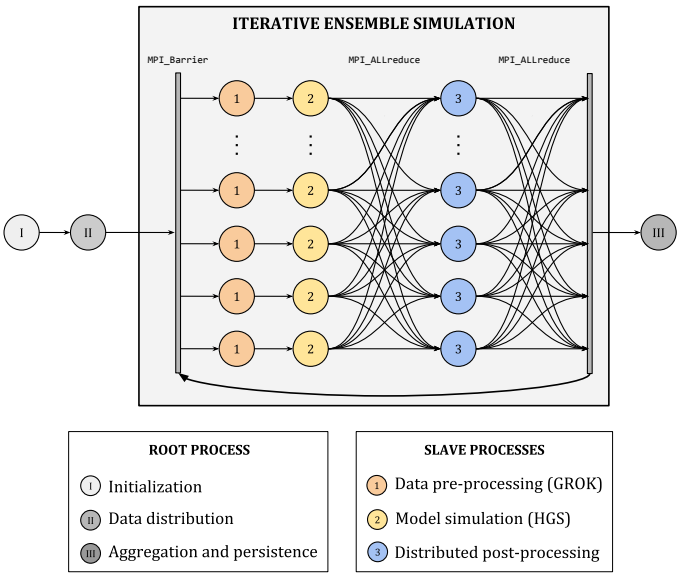


Figure 3: Original workflow of the MPI-based implementation.

The procedures executed in the MPI root process are identified using Roman numerals (from I to III), while numbers (1 to 3) refer to tasks that are computed distributively in MPI slave processes. There are three main stages in the workflow that comprise ensemble preparation, the iterative process of simulating the ensemble, and the final treatment of the results after the simulation. These stages are matched with the procedures in the figure as follows:

Ensemble preparation stage

I. Initialization

At the beginning of the workflow, the root MPI process initializes global data structures and reads the provided model parameters from the input files.

II. Data distribution

According to the initial model parameters, the root MPI process generates input files for each model realization and stores the files in separate directories on the network storage.

Iterative ensemble simulation

1. Data pre-processing (GROK)

After all running MPI processes reach the first synchronization point (a MPI barrier), each process execute the preprocessor GROK on the corresponding input directory in order to generate HGS-specific input files that will be written in the network file system.

2. Model simulation (HGS)

HGS reads the output of GROK, runs the model realization and writes the output to the network storage. At this point, all MPI processes synchronize for the second time using a MPI all-reduce directive, since further data updates require simulation results of all model realizations. This is a very network-intensive and memory-consuming stage due to the replication of the main matrix.

3. Distributed post-processing

During the data update process, each model realization is optimally weighted and updated with the most recent field measurements in order to reduce the simulation error. Each process is in charge of updating its block, and results are afterwards merged through another MPI all-reduce call.

Output management

III. Aggregation and persistence

After all iterations are completed and all MPI processes reach the barrier, the root MPI process aggregates the model simulation data from the realization directories and updates the global data structures. Before the program terminates, the root MPI process writes the model simulation results to the output files.

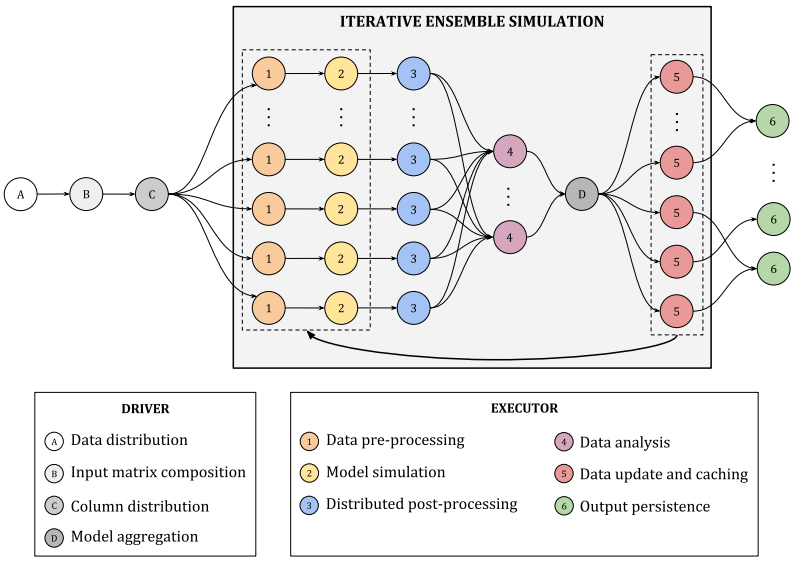


Figure 4: Final workflow of the adapted EnKF-HGS, matching stages 1, 2 and 3 with Fig. 3. Dashed lines indicate that the iteration is executed over a distributed dataset.

Applications with such execution model perform reasonably well in regular HPC environments due to certain important characteristics of the latter (e.g., availability of a high performance network storage, a low-latency broadband network connection). However, when moved to the cloud environment, the application performance might drop drastically (as it is illustrated in Subsection 5.2). In order to be able to benefit from the advantages of clouds while maintaining the performance at an acceptable level, certain modifications of the workflow are necessary.

4 Shifting to a Big Data Paradigm

As described, the original workflow consisted of a MPI implementation of the ensemble Kalman filter, which relied on two legacy binaries to execute the simulation (GROK and HGS). EnKF-HGS operates with a set of realizations, which constitute independent instantiations of the model with different parameters. They are simulated independently and the output is gathered afterwards for further processing.

Since the workflow is iterative, we selected the popular data analysis tool Spark as representative of a BD programming model and execution engine. Figure 4 depicts the stages that belong to the final implementation of the workflow in Spark. The procedures executed in the Spark driver process are identified using letters (from A to D), while numbers (1 to 6) refer to tasks that are computed distributively in the Spark executors. Similarly as in Fig. 3, the most relevant design and implementation details are described in the following paragraphs while matching these procedures to the core stages in the workflow.

Ensemble preparation stage

A. Data distribution

The first step is to load the necessary auxiliary files that every executor will need to properly run its data partition. This includes, for instance, the kernel binaries. Spark guarantees that these files will be available for the worker nodes in their current working directory.

B. Input matrix composition

Input data is read in the driver process in order to initialize the base model, composed of two main matrices, M_1 and M_2, in which each column $c_{1,r}$ and c_{2_r} corresponds to an instantiation, r, of the model. Additional data structures are created and initialised, and the parameters of the simulation are obtained.

C. Column distribution

Both matrices are distributed by columns in order to build the realization set, R. Each realization r is composed of the corresponding columns from both matrices, $c_{1,r}$ and $c_{2,r}$. Figure 5 depicts the realization data distribution process, which yields the distributed dataset that will be transformed in the following stages and iterations. The rationale behind distributing the workload this way is that each realization can be simulated independently from the others, without any further communication. Additionally, we forced each partition to hold the data for a single realization in order to induce fine-grained parallelism.

Iterative ensemble simulation

1. Data pre-processing (GROK)

After realizations are distributed, the GROK kernel writes the realization input data to a local file. HGS will read the realizations from these file in order to conduct the simulation of the model.

2. Model simulation (HGS)

With the input files from GROK, HGS simulates the model and writes its output for subsequent analysis. In steps 1 and 2 we must ensure that both binaries will be executed in the same node to exploit data locality. To achieve this, we run GROK and HGS in the same map function, which is an indivisible task in Spark. They thus act as an inner pipeline within the workflow.

3. Distributed post-processing

The post-processing stage is partially distributed. First, the output from each HGS execution is read in each executor in order to create an updated realization set, R'. With this information we create a distributed matrix M_1', and conduct several distributed operations to avoid gathering the whole matrix in the driver.

4. Data analysis

Further operations with auxiliary matrices are executed in the driver in order to filter and randomize the input for the following iteration. The goal of this stage is to minimise the size of the dataset that needs to be collected in the driver prior the model update. Note that to achieve this, significant data shuffles must be executed.

D. Data aggregation

In order to minimise data transfers, we exchange the original all-reduce MPI pattern for a the previous partial distributed computations plus this stage in which we collect and aggregate data that will be used to compute an update matrix. This update matrix is distributed afterwards, so that we can update the realizations without gathering the whole dataset in a single node.

5. Data update and caching

The distributed update matrix is used to update every realization in parallel. The resulting realization set is persisted to the local storage of the nodes as a fault-tolerance mechanism, and the following iteration starts.

Output management

6. Output persistence

After every iteration is executed, the output is stored to HDFS. Unlike the MPI version of the workflow, the new solution can execute I/O in parallel, as every partition is stored independently.

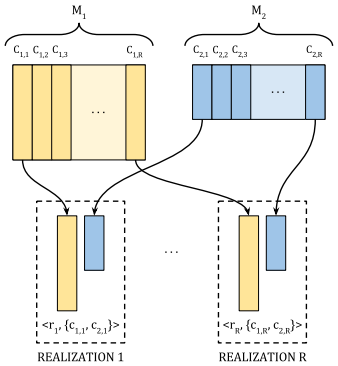


Figure 5: Column distribution procedure. Both matrices are split column-wise, and realizations are built with the corresponding column from both matrices.

Table 1: Technical specifications for the local cluster (testbed A) and the private cloud (testbed B).

Infrastructure	**Cluster**	**OpenNebula**
CPU	2 x Intel Xeon E5405 @2.00GHz	2 x Intel Xeon L5420 @2.50GHz
Total cores	8	8
Memory	8GB	8GB
OS	Linux Ubuntu 14.04.1 LTS	Linux Ubuntu LTS 14.0.4
Storage	2 x HD 1000GB + GlusterFS 3.6.9	HD 500GB
Network	1Gb/s Ethernet	1Gb/s Ethernet

Table 2: AmazonEC2 instance selection for the virtual clusters running the Spark and MPI platforms.

Platform	**Spark**			**MPI**	
Node role	**master**	**driver**	**slave**	**compute**	**storage**
Type	m3.medium	r3.xlarge	c4.2xlarge	c4.2xlarge	c4.large
Amount	1	1	16	16	3

5 Evaluation

The goal of our evaluation was to assess the benefits and drawbacks of the application of the techniques discussed in Sec. 2. We focused on absolute execution time and speed-up to analyse the effects of the memory and virtualization overheads of Spark and clouds, respectively. In addition, we conducted further experiments to evaluate the influence of I/O saturation on the application performance, which might occur due to a lack of high-speed network connection between distributed resources.

5.1 Experimental Setup

In order to assess the performance and scalability of the application, we selected three different execution infrastructures: a cluster, a private cloud running OpenNebula, and a virtual cluster on the AmazonEC2 public cloud. The specifications and limitations of these testbeds are described as follows.

Table 3: Technical specifications of the selected public cloud instances (testbed C), as provided by the AmazonEC2 documentation.

Type	Processor(*)	vCPU	Memory (GiB)	Storage (GB)	Network Performance
m3.medium	Intel Xeon E5-2670 v2 @2.5GHz Intel Xeon E5-2670 @2.6GHz	1	3.75	SSD	Moderate
r3.xlarge	Intel Xeon E5-2670 v2 @2.5GHz	4	30.5	SSD	Moderate
c4.2xlarge	Intel Xeon E5-2666 v3 @2.9GHz	8	15	EBS	High
c4.large	Intel Xeon E5-2666 v3 @2.9GHz	2	3.75	EBS	Moderate

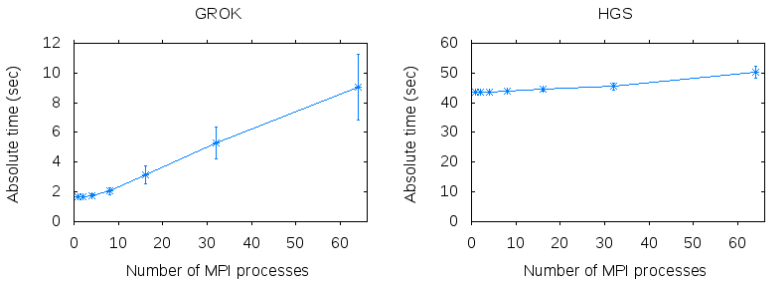
(*) More than one item is listed if VMs can be indistinctively launched on different physical processors.

Testbed A: Local Cluster This infrastructure comprised 11 slave nodes, with the specifications shown in Tab. 1. Each slave node holds 8GB of RAM and two Intel Xeon E5405 @2.00GHz processors, with four cores each. In addition, an auxiliary node was necessary to host the driver process of the Spark implementation, which required 7GB in the largest experiment we conducted. This means that the container in charge of running the driver would require 7GB plus a 10% memory overhead (as configured by default in the platform), 512MB extra memory for heap space, and other overhead sources like serialization buffers. Since Spark adds significant memory overhead to drivers and executors, we had to add a larger node to bypass the memory constraints in the slave nodes. As a consequence, we added a node with an overall amount of 94GB of RAM and four Intel Xeon E7-4807 @1.87GHz processors with six cores each. The local cluster had already a pre-installed network file system GlusterFS (a scalable and production-ready network file system[3]), which was necessary for the MPI implementation execution.

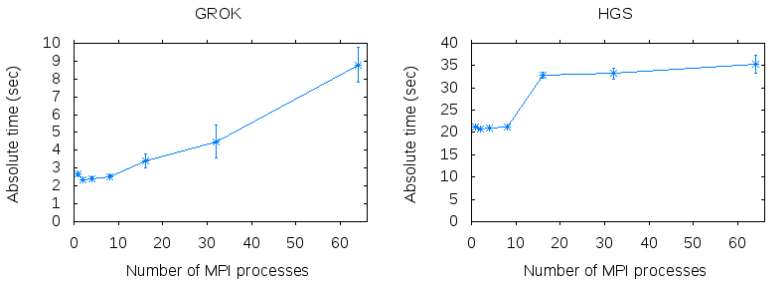
Testbed B: Private Cloud We relied on a virtual cluster running OpenNebula with the hardware described in Tab. 1. Notice that the main difference between this infrastructure and the cluster is the clock speed, which would benefit this testbed in the evaluation. To build the virtual cluster, we spawned 32 8-core VMs with 7.5GB of RAM each, the maximum available memory per VM. Notice that this memory limitation is relevant, as there was no workaround to fit the driver safely in the largest experiments. For the network file system, we deployed the latest version of GlusterFS on 3 additional storage nodes, which were organized in a distributed volume with no data-replication in order to maximize the storage performance. On the computing nodes side, we exploited the FUSE-based Gluster Native Client for highly concurrent access to the file system.

Testbed C: Public Cloud We selected *c4.2xlarge* instances as slaves due to their balance between number of cores and amount of memory, and a *r3.xlarge* instance with larger memory to hold the driver. The virtual cluster for Spark was composed of 16 workers (128 cores in total), an *m3.medium* master, and the additional dedicated VM for the driver. MPI ran on a virtual cluster built with identical slaves, plus three additional *c4.large* storage nodes running GlusterFS with the configuration similar to the previous testbed. Each storage node was provisioned with an 5GB general purpose SSD brick. Table 2 shows a summary of the selected instances and their assigned roles in both the Spark and MPI execution platforms. In addition, Tab. 3 describes the hardware characteristics published by the provider[4] [48]. Given its public nature, this testbed could yield the largest differences in performance among independent executions. To assess this possibility, we conducted five runs of both applications for a small example comprising eight realizations and eight cores, and detected that the standard deviation is never higher than 5% of the average execution time.

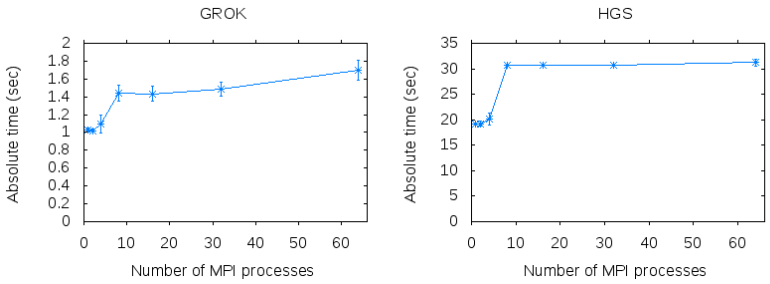
[3]GlusterFS is available at https://www.gluster.org/
[4]Retrieved from https://aws.amazon.com/ec2/instance-types/



(a) Execution time for GlusterFS on cluster.



(b) Execution time for EFS on AmazonEC2.



(c) Execution time for GlusterFS on AmazonEC2.

Figure 6: Execution time for the GROK and HGS kernels running on different infrastructures and file systems.

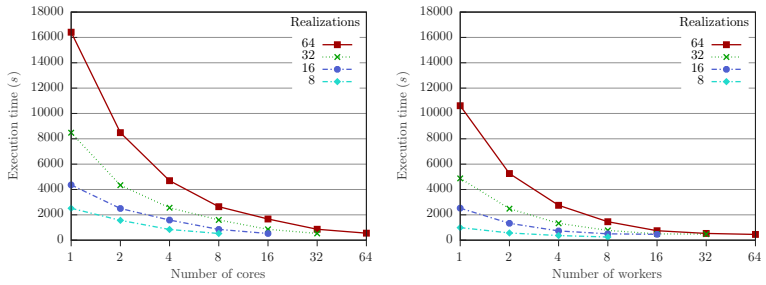
5.2 I/O Saturation per Infrastructure

Before evaluating the scalability of two implementations and comparing their performance, it is important to measure the effect of concurrent access to a network storage on execution time of the overall application. As described in subsection 3.2, the forward propagation phase of the EnKF-HGS workflow runs an ensemble of concurrent HGS model realizations, which use a file-based data exchange mechanism via a network file system. Unlike the Spark implementation, where each execution runs on local data by design.

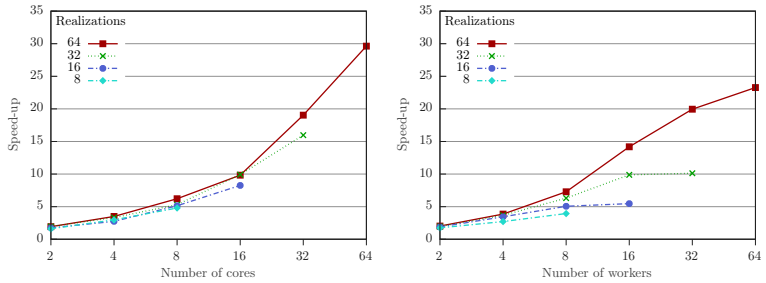
One interesting but counter-intuitive aspect we noticed in this evaluation is that Spark runs the experiments faster, at least until the scalability starts degrading. We believed this was due to the impact of the concurrent accesses to the network storage device conducted by the parallel kernel executions. Indeed, in the case of Spark, each execution runs on local data by design, which might be behind these performance differences.

In order to measure this effect, we had to guarantee that the number of I/O operations was identical for every instance of the GROK and HGS kernels. Since both kernel binaries are pre-built black-boxes, we modified the EnKF-HGS to compute one model realization N times, where N is the number of ensemble members. Thus, instead of computing N different model realizations, which most likely would result in a different number of the I/O operations for each individual realization, we ensured that each instance of the simulator performed the same I/O operations by making each input identical.

We have measured the relation between the number of concurrent processes (i.e., GROK and HGS kernels) that are using the network file system and the execution time in two opposed testbeds: (i) local cluster, and (ii) public cloud, which were described above. Additionally, for the public



(a) Execution time (MPI). (b) Execution time (Spark).



(c) Speed-up (MPI). (d) Speed-up (Spark).

Figure 7: Execution time and speed-up for the MPI and Spark workflows, running on a local cluster.

cloud testbed, we repeated the experiment with the Elastic File System (EFS), which is a native file system for AmazonEC2 environment. For the experiment's execution, we modified EnKF-HGS to run 1 to 64 MPI processes, where each process was provided with one CPU core.

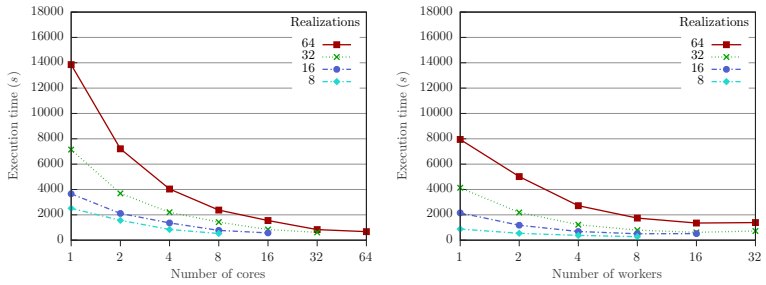
Figure 6 shows the impact of concurrent accesses to network storage on the execution time of the kernel binaries (GROK and HGS separately). Surprisingly, even in (a), which is considered as an "MPI-friendly" environment, the performance of the I/O intensive kernel (GROK) drops 5 times in case of 64 concurrently running instances of the kernel. The results in (b) and (c) illustrate that in cloud environment the situation is even more dramatic as a still significant performance drop happens not only for the I/O intensive kernel, but also for the compute intensive one. From this experiment we can clearly see that a typical MPI-based application assumptions like availability of a high performance network file system and a low-latency broadband network connection should be taken extremely serious as a violation of such assumptions might lead to a tremendous application performance drop. In this experiment, we artificially equalized the number of the concurrent I/O operations, which resulted in a large performance drop. For a realistic ensemble, consisting of individual model realizations, the computations most likely would not last the same, which in turn would reduce the number of concurrent I/O operations and thus reducing the saturation.

The following experiments will illustrate the performance and scalability of two implementations in the presented testbeds.

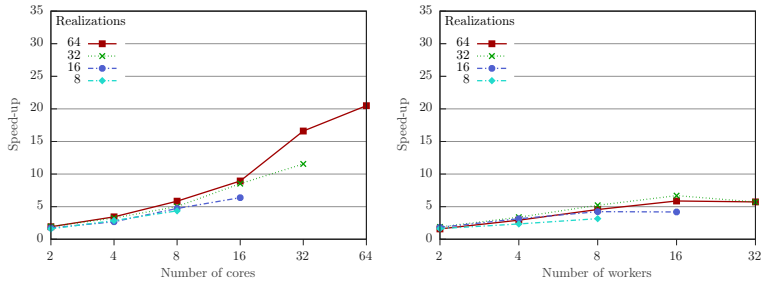
5.3 Execution Models: Spark vs. MPI

The objective of these experiments is to detect the effects the execution models have on the performance of the workflow execution. We analysed the MPI-based implementation of EnKF-HGS, and its data-centric version built in Spark, both running in the local cluster formerly described.

We allocated Spark executors with one core in order to fairly compare scalability against single-core MPI processes. We experimented with increasing realization volumes and executor number, we measured the absolute execution time for a single execution (including the job launch time required by Spark), and we computed speed-up. The results for this execution on a local cluster are shown in Fig. 7, in which (a) and (c) correspond to MPI, and (b) and (d) correspond to Spark. Remarkably, Spark yields better execution times for every experiment, and its speed-up is better the larger is the experiment for a given number of workers. This might be a result of the redesign process. However, the speed-up in Spark for the largest experiment (i.e. 64 realizations on 64 executors) is lower than in the MPI case. The problem in this case is that the 64 executors cannot be scheduled at once due to their large memory requirements.



(a) Execution time (MPI). (b) Execution time (Spark).



(c) Speed-up (MPI). (d) Speed-up (Spark).

Figure 8: Execution time and speed-up for the MPI and Spark workflows, running on a private OpenNebula cloud.

The main conclusion from this experiment is that, while the BD-inspired approach shows surprising performance results, the memory overhead of the execution framework hurts scalability, as less parallel executors can be allocated in the same infrastructure. As a result, the slimmer MPI processes seem more suitable for large scale execution of this workflow. Another interesting aspect is related to the post-processing stage of the workflow and its effect on the overall execution time. At some point, with the growing number of the parallel executors, the post-processing computation becomes shorter in time than the data transferring time. As a result, the post-processing stage starts to affect the overall execution time more than with a fewer number of the parallel executors.

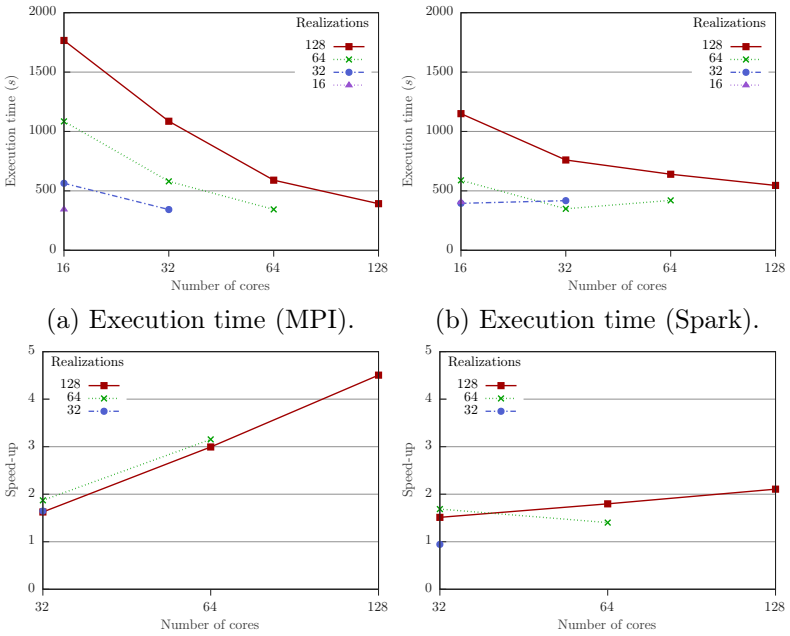
Another major drawback we found in Spark is the limited capability of the framework to establish simultaneous connections among many nodes. Our first approach to make data integration was to use memory mechanisms instead of files. However, it requested to create an all-to-all communication pattern. Even if this pattern is straightforward and very efficient in MPI, we were unable to scale this approach up to 256 nodes in Spark, as the RPC system underlying Spark failed due to the impossibility to create connections.

5.4 Computing Models: Cluster vs. Private Cloud

After analysing the programming models, we focused on the underlying computing models to assess their impact in performance. Hence, we conducted further experiments on the OpenNebula private cloud with the Spark and MPI implementations.

Besides the cluster results shown in the previous experiments (Fig. 7), Fig. 8 reflects the same metrics for the experiments conducted in OpenNebula for a single execution. Regarding the results for Spark, which is the approach that should benefit the most from BD-oriented environments like clouds, while the overall evolution of speed-up seems similar in relative terms, we can clearly see that in OpenNebula the results are much more extreme, with even lower execution times, but also smaller speed-ups. Interestingly, MPI also shows a similar behaviour, with lower execution times and degraded speed-ups. These results could be related to the larger frequency in the physical processors of the virtual cluster, which yields faster executions, and the lower memory per core ratio, which hurts scalability.

With respect to the results for Spark, there are two remarkable exceptions. The first one is the 64 realization execution with 64 workers, which failed as the container corresponding to the driver violates the system's memory limit, hence this result is not included. The other one is that the result for 32 realizations with 32 workers shows the same result as the 16-worker one. We noticed that this is due to one –two, at most– executors missing their heartbeat in the driver, which caused



(a) Execution time (MPI). (b) Execution time (Spark).

(c) Speed-up (MPI). (d) Speed-up (Spark).

Figure 9: Execution time and speed-up for the MPI and Spark workflows, running on a virtual cluster on the AmazonEC2 cloud.

their assigned task to time-out. Under these circumstances, Spark considers the worker is dead, hence reschedules the task in a different node. Since there are no more resources to launch a new container, it has to wait for an executor to finish, and then launch the former tasks. Considering that there is a synchronization point after the concurrent computation of the realizations, the result is that the execution time is doubled, thus matching the time obtained for 16 executors.

6 Lessons Learned

Our previous works showed that BD programming models –like Hadoop and Spark– and infrastructures –such as clouds– could be used to improve the efficiency and scalability of some types of scientific applications with minor modifications. More specifically, we detected that simulators relying on parameter-sweep and partitionable domains, and kernel-based workflows comprising many, loosely-coupled tasks could greatly benefit from the massive parallelism of BD paradigms. The evaluation and literature analysis in this work indicate that these techniques are still far from disrupting the vast set of Monte Carlo workflows, in particular as the scale increases. This experience and the current state of the literature is summarized in Tab. 4 and detailed below.

We used Spark, a framework specially built for iterative processes, yet the iterative workflow we implemented was not able to scale properly, mainly due to memory overhead and bottlenecks introduced by the execution environment, as present in other Java-based MapReduce platforms [43] and Spark itself [2]. The deep component stack and its dependence on the JVM yield a significant memory consumption that also affects execution time due to frequent garbage collection operations [20] and serialization if bindings to other languages are used [41]. Moreover, we found a critical limitation in the scalability of the RPC communication system in Spark to establish all-to-all communication patterns.

Although Spark is perceived as a fault-tolerant platform, we found that tasks associated with containers killed by the resource manager were not rescheduled, hence data was lost, and the whole execution eventually died. However, task rescheduling does happen in the case of network related failures.

Another observation is the rough fitness of the simulation ensemble paradigm to the target use cases of the language, which yielded suboptimal performance as it is not possible to attain an implementation that exploits the full potential of both the Spark platform and the ensemble algorithm. This is mainly due to the limitations on matrix operations and data collection stages.

Table 4: Summary of the main features detected during the evaluation and literature review for the BD and the HPC paradigms.

	Big Data paradigm
Pros	Fault-tolerance by design
	Transparent data locality
	Job and task scheduling at platform level
Cons	Low control of resource management
	Significant memory overhead
	Poor integration with kernels
	Key-value only
	Deep software and communication stack
	HPC paradigm
Pros	Low resource consumption
	Efficient communication
	Generalist and tailorable processes
Cons	Limited parallel abstractions
	No native provenance or replication
	Steep learning curve

However, the scalability issues we found in Spark for this use case are still valid, since they are tightly related to its overhead.

The workflow relies on two third-party kernels, acting as black-boxes with data schemes and proprietary code. This, in addition to the fact that we based our implementation in the current MPI version of the workflow, might lead to an implementation that does not exploit the full potential of the Spark platform. Further parallelizations on the post-processing stage could be feasible, but it would require a full re-engineering of the workflow to the limitations on minimize the matrix operation limitations and to reduce the size of collected data. All of these aspects could hurt the behaviour of the BD-based implementation against the solution built for MPI. However, the scalability issues we found in Spark for this use case are still valid, since they are mainly related with its overhead.

Despite the former, we experienced that building a solution with Spark, and for a consolidated cloud like Amazon EC2, definitely reduced development time given the abstract nature of data and objects in analytics. We were able to tailor the infrastructure as desired, without worrying about other users or software compatibility issues. We only had to consider the design of the workflow, and some specific implementation particularities of the language and the available functionalities. The data-centric nature of Spark allowed us to exploit the parallelism of the workflow, with good experimental results as shown in this work. Due to the scale of our experiments we could not detect network issues in MPI. However, we found that I/O management could become a bottleneck. Being able to distribute data and access it locally with Spark was key to achieve a competitive workflow, at least at small scale.

Another main benefit of having Spark as execution engine was its underlying resource manager and distributed file system, which had a major impact in easing data distribution and task management. However, it was necessary to conduct a very time-consuming tailoring of the configuration given its outstanding impact in the final performance and stability [11].

There is another major problem we found with BD-oriented frameworks, which is the problems they are built to solve. Some BD-oriented frameworks show drawbacks in terms of generality and versatility. For example, our major development issue was dealing with operations involving the large matrices needed in the post-processing stage of the workflow. Although there are libraries to handle distributed matrices, like Spark's MLLib, the functionality is limited to the common

operations needed for data analysis. Hence, we had to create ad-hoc workarounds to implement some matrix-dependant sections of the workflow, which degraded our development experience.

Finally, it could be claimed that the authors could be biased given our previous works. However, our experience with significant positive results for BD approaches allowed us to remain critic with our new results. Our approach in this work was purely experimental, thus we highlighted every relevant issue we found in the development and evaluation stages, in order to detect new research opportunities.

7 Conclusions

Scientific workflows are becoming data-intensive, and their scale continues to grow with larger data volumes and problem complexities. There is a rising interest in exploiting the opportunity to leverage Big Data application models and infrastructures to increase workflow scalability. Nevertheless, the benefits and drawbacks of these new tools have not been fully studied yet.

This work has explored the effects that those paradigms could have in current workflows. We aimed to detect their benefits and drawbacks, and to extract from these knowledge a series of lessons and future research topics. We focused this work in two key elements in the Big Data ecosystems: cloud computing, and data analysis programming models. As representatives of these paradigms, we selected Amazon EC2, a popular public cloud; OpenNebula, a private cloud management system; and Apache Spark, one of the most active data analytics frameworks. We experimented with a specific workflow from the hydrology domain, and analysed its performance and scalability.

From the infrastructure side, we have seen that memory has become the limiting factor for new BD frameworks like Spark. As a consequence, instead of tailoring the hardware to the execution of many small tasks, upcoming data-intensive infrastructures should heavily invest in both volatile and non-volatile memory. These additional resources in the memory stack could mitigate the requirements of new workloads, and they would help the support of the emerging in-memory and caching mechanisms coming from data-aware computing.

Regarding workflow development and deployment, we conclude that a promising research line for large scale scientific workflows would be working towards an hybrid approach between MPI and Big Data-oriented data abstractions. Such model would blend the slim MPI processes and their generalist nature, with the ability to reason about data processing without explicitly implementing data parallelism that BD frameworks provide. This would result in a highly productive and efficient mechanism to build and deploy both scientific workflows and Big Data applications.

The first steps of our experimentation were conducted in our limited private infrastructures, hence the difficulty to conduct a large-scale evaluation. Despite this issue, there is a key positive aspect in this, which is that our infrastructures are representative of the capabilities of many small and medium sized research groups in science and data analysis. Moreover, this scale was sufficient to detect major scalability issues, and we could contrast this when attempting to run larger experiments in the public cloud. This should be complemented with full large-scale experimentations, which we found infrequent in the literature.

While this use case was relevant for our objectives in terms of complexity and size, further experimentation with other workflows built with different patterns would be necessary to corroborate our results and conclusions. We believe that an extensive memory consumption analysis would be required to fully understand how memory is used within the Spark platform. This is left for future work, as it is key to understand how a slimmer framework could be built. In addition, further analysis of the cost-performance trade-off of applying Big Data paradigms to workflows should be conducted, as well as a detailed analysis of the I/O-related overhead specific to the workflows.

Acknowledgements

This work was supported by the Spanish Ministry of Economics and Competitiveness grant TIN-2013-41350-P, the IC1305 COST Action "Network for Sustainable Ultrascale Computing Platforms" (NESUS), and the FPU Training Program for Academic and Teaching Staff Grant FPU15/00422 by the Spanish Ministry of Education. We gratefully acknowledge Wolfgang Kurtz (IBG-3, Forschungszentrum Jülich GmbH) for providing the EnKF-HGS simulator source code; and Oliver Schilling (CHYN, University of Neuchâtel) for providing the hydrological model use-case.

References

[1] Ananthanarayanan, R., Gupta, K., Pandey, P., Pucha, H., Sarkar, P., Shah, M., Tewari, R.: Cloud analytics: Do we really need to reinvent the storage stack? In: HotCloud (2009)

[2] Awan, A.J., Brorsson, M., Vlassov, V., Ayguade, E.: How Data Volume Affects Spark Based Data Analytics on a Scale-up Server, pp. 81–92. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-29006-5_7

[3] Berriman, G.B., Deelman, E., Juve, G., Rynge, M., Vöckler, J.S.: The application of cloud computing to scientific workflows: a study of cost and performance. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 371(1983) (2012)

[4] Bobashev, G.V., Goedecke, D.M., Yu, F., Epstein, J.M.: A hybrid epidemic model: combining the advantages of agent-based and equation-based approaches. In: Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come. pp. 1532–1537. IEEE Press (2007)

[5] Brunner, P., Simmons, C.T.: Hydrogeosphere: A fully integrated, physically based hydrological model. Ground Water 50(2), 170–176 (2012)

[6] Burgers, G., van Leeuwen, P.J., Evensen, G.: Analysis scheme in the ensemble Kalman filter. Monthly Weather Review 126(6), 1719–1724 (1998)

[7] Bux, M., Brandt, J., Lipka, C., Hakimzadeh, K., Dowling, J., Leser, U.: Saasfee: Scalable scientific workflow execution engine. Proc. VLDB Endow. 8(12), 1892–1895 (Aug 2015)

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

[9] Callaghan, S., Maechling, P., Small, P., Milner, K., Juve, G., Jordan, T.H., Deelman, E., Mehta, G., Vahi, K., Gunter, D., Beattie, K., Brooks, C.: Metrics for heterogeneous scientific workflows: A case study of an earthquake science application. Int. J. High Perform. Comput. Appl. 25(3), 274–285 (Aug 2011)

[10] Caíno-Lores, S., Lapin, A., Kropf, P.G., Carretero, J.: Lessons Learned from Applying Big Data Paradigms to Large Scale Scientific Workflows. In: WORKS@ SC. pp. 54–58 (2016), https://pdfs.semanticscholar.org/a985/f4804d342b0f9709486ef7b1e7b5b1b42cac.pdf

[11] Chaimov, N., Malony, A., Canon, S., Iancu, C., Ibrahim, K.Z., Srinivasan, J.: Scaling spark on hpc systems. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing. pp. 97–110. HPDC '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2907294.2907310

[12] Chiang, G.T., Dove, M.T., Bovolo, C.I., Ewen, J.: Implementing a grid/cloud escience infrastructure for hydrological sciences. In: Guide to e-Science, pp. 3–28. Springer (2011)

[13] Dede, E., Govindaraju, M., Gunter, D., Ramakrishnan, L.: Riding the elephant: Managing ensembles with hadoop. In: Proceedings of the 2011 ACM International Workshop on Many Task Computing on Grids and Supercomputers. pp. 49–58. MTAGS '11, ACM, New York, NY, USA (2011)

[14] Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: The montage example. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. pp. 50:1–50:12. SC '08, IEEE Press, Piscataway, NJ, USA (2008)

[15] Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: The montage example. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. pp. 50:1–50:12. SC '08, IEEE Press, Piscataway, NJ, USA (2008)

[16] Duro, F., García, J., Isaila, F., Carretero, J., Wozniak, J., R., R.: Flexible data-aware scheduling for workflows over an in-memory object store. In: Proceedings of IEEE/ACM CCGrid 2016 (2016)

[17] Etemadpour, R., Bomhoff, M., Lyons, E., Murray, P., Forbes, A.: Designing and evaluating scientific workflows for big data interactions. In: Big Data Visual Analytics (BDVA), 2015. pp. 1–8 (Sept 2015)

[18] Evangelinos, C., Hill, C.: Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. ratio 2(2.40), 2–34 (2008)

[19] Evensen, G.: Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. Journal of Geophysical Research: Oceans 99(C5), 10143–10162 (1994)

[20] Gog, I., Giceva, J., Schwarzkopf, M., Vaswani, K., Vytiniotis, D., Ramalingan, G., Costa, M., Murray, D., Hand, S., Isard, M.: Broom: Sweeping out garbage collection from big data systems. Young 4, 8 (2015)

[21] Gupta, A., Milojicic, D.: Evaluation of hpc applications on cloud. In: 2011 Sixth Open Cirrus Summit. pp. 22–26 (Oct 2011)

[22] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the use of cloud computing for scientific workflows. In: eScience, 2008. eScience '08. IEEE Fourth International Conference on. pp. 640–645 (Dec 2008)

[23] Jain, A., Ong, S.P., Hautier, G., Chen, W., Richards, W.D., Dacek, S., Cholia, S., Gunter, D., Skinner, D., Ceder, G., Persson, K.A.: Commentary: The materials project: A materials genome approach to accelerating materials innovation. APL Mater. 1(1), 011002 (2013)

[24] Jyrkama, M.I.: A methodology for estimating groundwater recharge, vol. 65 (2004)

[25] Kurtz, W., Hendricks Franssen, H.J., Kaiser, H.P., Vereecken, H.: Joint assimilation of piezometric heads and groundwater temperatures for improved modeling of river-aquifer interactions. Water Resources Research 50(2), 1665–1688 (2014)

[26] Lapin, A., Schiller, E., Kropf, P., Schilling, O., Brunner, P., Kapic, A.J., Braun, T., Maffioletti, S.: Real-time environmental monitoring for cloud-based hydrogeological modeling with hydrogeosphere. In: High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on. pp. 959–965 (Aug 2014)

[27] Lin, G., Han, B., Yin, J., Gorton, I.: Exploring cloud computing for large-scale scientific applications. In: 2013 IEEE Ninth World Congress on Services. pp. 37–43 (June 2013)

[28] Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. Journal of Grid Computing 13(4), 457–493 (2015)

[29] Lu, S., Li, R.M., Tjhi, W.C., Lee, K.K., Wang, L., Li, X., Ma, D.: A framework for cloud-based large-scale data analytics and visualization: Case study on multiscale climate data. In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. pp. 618–622. IEEE (2011)

[30] Luckow, A., Mantha, P., Jha, S.: Pilot-abstraction: A valid abstraction for data-intensive applications on hpc, hadoop and cloud infrastructures? arXiv preprint arXiv:1501.05041 (2015)

[31] Maltzahn, C., Molina-Estolano, E., Khurana, A., Nelson, A.J., Brandt, S.A., Weil, S.: Ceph as a scalable alternative to the hadoop distributed file system. login: The USENIX Magazine 35, 38–49 (2010)

[32] Marcu, O.C., Costan, A., Antoniu, G., Pérez-Hernández, M.S.: Spark versus flink: Understanding performance in big data analytics frameworks. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER). pp. 433–442 (Sept 2016)

[33] Maxwell, R.M., Putti, M., Meyerhoff, S., Delfs, J.O., Ferguson, I.M., Ivanov, V., Kim, J., Kolditz, O., Kollet, S.J., Kumar, M., Lopez, S., Niu, J., Paniconi, C., Park, Y.J., Phanikumar, M.S., Shen, C., Sudicky, E.A., Sulis, M.: Surface-subsurface model intercomparison: A first set of benchmark results to diagnose integrated hydrology and feedbacks. Water Resources Research 50(2), 1531–1549 (2014)

[34] Mell, P., Grance, T.: Effectively and securely using the cloud computing paradigm. NIST, Information Technology Laboratory pp. 304–311 (2009)

[35] Mork, R., Martin, P., Zhao, Z.: Contemporary challenges for data-intensive scientific workflow management systems. In: Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science. pp. 4:1–4:11. WORKS '15, ACM, New York, NY, USA (2015)

[36] de Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M.: Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In: 2010 IEEE 3rd International Conference on Cloud Computing. pp. 378–385 (July 2010)

[37] Partington, D., Brunner, P., Frei, S., Simmons, C.T., Werner, A.D., Therrien, R., Maier, H.R., Dandy, G.C., Fleckenstein, J.H.: Interpreting streamflow generation mechanisms from integrated surface-subsurface flow models of a riparian wetland and catchment. Water Resources Research 49(9), 5501–5519 (2013)

[38] Parunak, H.V.D., Savit, R., Riolo, R.L.: Agent-based modeling vs. equation-based modeling: A case study and users' guide. In: International Workshop on Multi-Agent Systems and Agent-Based Simulation. pp. 10–25. Springer (1998)

[39] Raicu, I., Foster, I., Zhao, Y.: Many-task computing for grids and supercomputers. In: Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on. pp. 1–11 (Nov 2008)

[40] Reed, D.A., Dongarra, J.: Exascale computing and big data. Communications of the ACM 58(7), 56–68 (2015)

[41] Salucci, L., Bonetta, D., Binder, W.: Lightweight multi-language bindings for apache spark. In: Proceedings of the 22Nd International Conference on Euro-Par 2016: Parallel Processing - Volume 9833. pp. 281–292. Springer-Verlag New York, Inc., New York, NY, USA (2016), http://dx.doi.org/10.1007/978-3-319-43659-3_21

[42] Schilling, O., Doherty, J., Kinzelbach, W., Wang, H., Yang, P., Brunner, P.: Using tree ring data as a proxy for transpiration to reduce predictive uncertainty of a model simulating groundwater–surface water–vegetation interactions. Journal of Hydrology 519, Part B, 2258 – 2271 (2014)

[43] Shi, X., Chen, M., He, L., Xie, X., Lu, L., Jin, H., Chen, Y., Wu, S.: Mammoth: Gearing hadoop towards memory-intensive mapreduce applications. IEEE Transactions on Parallel and Distributed Systems 26(8), 2300–2315 (Aug 2015)

[44] Swinerd, C., McNaught, K.R.: Design classes for hybrid simulations involving agent-based and system dynamics models. Simulation Modelling Practice and Theory 25, 118–133 (2012)

[45] Szabo, C., Sheng, Q.Z., Kroeger, T., Zhang, Y., Yu, J.: Science in the cloud: Allocation and execution of data-intensive scientific workflows. Journal of Grid Computing 12(2), 245–264 (2014)

[46] Therrien, R., McLaren, R., Sudicky, E., Panday, S.: A Three-dimensional Numerical Model Describing Fully-integrated Subsurface and Surface Flow and Solute Transport. Tech. rep. (2010)

[47] Vahi, K., Rynge, M., Juve, G., Mayani, R., Deelman, E.: Rethinking data management for big data scientific workflows. In: Big Data, 2013 IEEE International Conference on. pp. 27–35 (Oct 2013)

[48] Wang, K., Khan, M.M.H.: Performance prediction for apache spark platform. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems. pp. 166–173 (Aug 2015)

[49] Wang, Y., Goldstone, R., Yu, W., Wang, T.: Characterization and optimization of memory-resident mapreduce on hpc systems. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium. pp. 799–808 (May 2014)

[50] Yang, C., Goodchild, M., Huang, Q., Nebert, D., Raskin, R., Xu, Y., Bambacus, M., Fay, D.: Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? International Journal of Digital Earth 4(4), 305–329 (2011)

[51] Zalta, E.N., et al.: Stanford encyclopedia of philosophy (2003)

[52] Zhang, H., Chen, G., Ooi, B.C., Tan, K.L., Zhang, M.: In-memory big data management and processing: A survey. IEEE Transactions on Knowledge and Data Engineering 27(7), 1920–1948 (July 2015)

[53] Zhang, Z., Barbary, K., Nothaft, F.A., Sparks, E., Zahn, O., Franklin, M.J., Patterson, D.A., Perlmutter, S.: Scientific computing meets big data technology: An astronomy use case. In: Big Data (Big Data), 2015 IEEE International Conference on. pp. 918–927 (Oct 2015)

[54] Zhang, Z.: Processing data-intensive work ows in the cloud (2012)

[55] Zhao, Y., Fei, X., Raicu, I., Lu, S.: Opportunities and challenges in running scientific work-flows on the cloud. In: Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on. pp. 455–462 (Oct 2011)

[56] Zhao, Y., Raicu, I., Foster, I.: Scientific workflow systems for 21st century, new bottle or new wine? In: 2008 IEEE Congress on Services - Part I. pp. 467–471 (July 2008)