Lessons Learned from Applying Big Data Paradigms to a Large Scale Scientific Workflow

Silvina Caíno-Lores
University Carlos III of Madrid
Av. Universidad, 30, Leganés,
Madrid, Spain
scaino@inf.uc3m.es

Andrei Lapin
University of Neuchâtel
Neuchâtel, Switzerland
andrei.lapin@unine.ch

Peter Kropf
University of Neuchâtel
Neuchâtel, Switzerland
peter.kropf@unine.ch

Jesús Carretero
University Carlos III of Madrid
Av. Universidad, 30, Leganés,
Madrid, Spain
jcarrete@inf.uc3m.es

ABSTRACT

The increasing amount 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 with combining the traditional high-performance computing and grid-based approaches for scientific workflows, with Big Data analytics paradigms. Our goal was to assess and discuss the suitability of such data-intensive-oriented mechanisms for production-ready workflows, especially in terms of scalability, focusing on a key element in the Big Data ecosystem: the data-centric programming model. Hence, we reproduced the functionality of a MPI-based iterative workflow from the hydrology domain, EnKF-HGS, using the Spark data analysis framework. We conducted experiments on a local cluster, and we relied on our results to discuss promising directions for further research.

CCS Concepts

•General and reference → Experimentation; •Software and its engineering → Data flow languages; Massively parallel systems; •Computing methodologies → *Simulation evaluation; Data assimilation;* **•Applied computing** → *Environmental sciences;*

Keywords

Scientific workflows; Big Data; Cloud Computing; Apache Spark; Hydrology

1. INTRODUCTION

Scientific workflows are key tools in many research areas that rely on multiple, diverse, and distributed operations over various datasets, usually yielding significant computational complexity and data dependencies. Nowadays, the increasing amount of input, intermediate, 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 [13], trends show that large scale workflows would have to address increasing data sizes, easily reaching peta-scale [18]. There has also been a rise of scientific many-task computing (MTC) workflows [4], which are capable to handle the huge data volume and the massive computational requirements of these simulations.

In this context, scientific workflows face new performance and scalability challenges in terms of data management, workload distribution, load balance, and scheduling, to name a few. 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 [9]. For example, typical BD programming models, such as Apache Hadoop, have been considered to substitute MPI parallelism induction mechanisms, following a data-centric approach.

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 [9]. For example, typical BD programming models, such as Apache Hadoop, have been considered to substitute MPI parallelism induction mechanisms, following a data-centric approach. Following this trend, BD paradigms are increasingly seen as alternatives to traditional HPC approaches for some major types of scientific applications, especially those with many loosely-coupled tasks [11], or heterogeneous tasks with few interdependences [12].

Our hypothesis is that BD techniques could be used to scale-up scientific workflows, although the architectural differences between the analytics and scientific worlds might require novel approaches to achieve satisfactory results. Therefore, in this work we aimed to assess the suitability of such data-intensive-oriented mechanisms for production-ready workflows, especially in terms of scalability.

In previous works [3] 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 as map-reduce-based wrap-

pers of the core implementation of their models.

In this paper we address the suitability of such data-intensive-oriented mechanisms for production-ready workflows. To study this, we reproduced the functionality of an iterative workflow from the hydrology domain, EnKF-HGS [8], in Apache Spark 1.6.0[1], which is currently a major representative of the data-centric analytics ecosystem. We compared the original workflow and the Spark implementation in a traditional cluster to evaluate scalability, and we 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 those paradigms, platforms and infrastructures currently used in BD, to a scientific workflow. The rest of this paper contains: background on the BD techniques we relied on and their relation with scientific workflows, and a high-level description of the workflow from the hydrology domain we considered as use case (Section 2); key architectural details of the proposed solution implemented in Spark (Section 3); the preliminary evaluations we conducted to study the behaviour of the resulting redesigned workflow (Section 4); and relevant highlights and directions for future work (Section 5).

2. BACKGROUND

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 [17]. 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 [15].

Data-centric analytics and BD tools like Hadoop and Spark are being explored to provide straightforward data distribution and caching mechanisms in pleasingly-parallel data-intensive HPC applications. The inherently parallel nature of these tools has resulted in positive experimental results showing their suitability for massively parallel workloads like MTC-like workflows [16]. Nevertheless, challenges remain with respect to workflows built with a pure HPC focus, like those described in [10], which rely on MPI and traditional storage infrastructures.

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 [5]. In this work, we focus not only in the design and deployment of BD-inspired scientific workflows, but also in the performance and scalability issues they inherit from the platform and infrastructure. We aim to provide insight on the potential benefits of redesigning HPC-oriented workflows to BD platforms, while reflecting the technical and performance issues that arise from these paradigms.

Major examples of these resource-intensive workflows are multi-scale data analysis applications. The hydrology domain is a representative example of the former. One of the state of the art simulators in this domain is the EnKF-HGS workflow [7]. EnKF-HGS is an iterative workflow that implements an MPI version of the ensemble Kalman filter (EnKF) technique for sequential data assimilation [6, 2]. EnKF-HGS runs an ensemble of model instantiations –which we call *realizations*– with different combinations of input parameters and initial conditions.

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

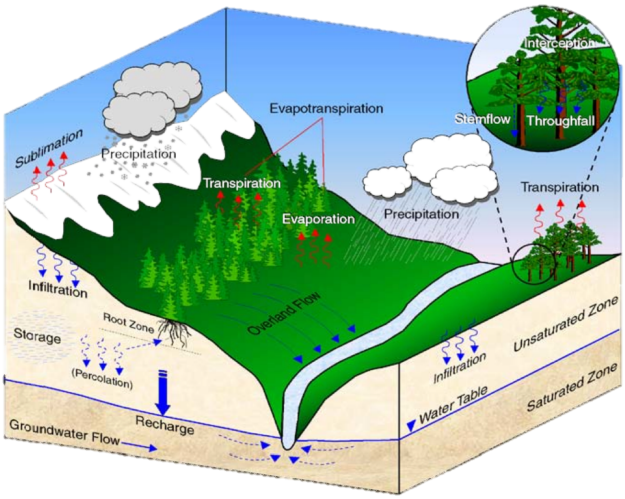


Figure 1: Typical surface water and groundwater processes in a pre-alpine type of valleys.

As shown in Figure 1, HGS allows the numerical simulation of all the relevant surface water and groundwater processes in a pre-alpine type of valleys. Hence, each HGS simulation in the ensemble of realizations represents a long-running compute-intensive process, which comprises the sequential execution of two proprietary simulation kernels: GROK and HydroGeoSphere (HGS) [14, 1]. 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 differential equations. With the HGS simulation results, each model realization is updated with the environmental field measurements and optimally weighted in order to achieve a higher quality model prediction.

3. SHIFTING TO A BIG DATA PARADIGM

As described, the original workflow consisted of an MPI implementation of an EnKF, 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, but 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. The most relevant design and implementation details of the final implementation of the workflow in Spark are described in the following paragraphs. 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.

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 $c_{2,r}$, so that a data distribution process is needed to create 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. After realizations are distributed, the followoing steps are executed for each realization:

1. Data pre-processing

The GROK kernel writes the realization input data to a local file. HGS will read the realizations from this file in order to conduct the simulation of the model.

2. Model simulation

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 a the driver prior the model update. Note that to achieve this, significant data shuffles must be executed.

Table 1: Technical specifications for the local cluster.

CPU	2 x Intel Xeon E5405 @2.00GHz
Total cores	8
Memory	8GB
OS	Linux Ubuntu 14.04.1 LTS
Storage	2 x HD 1000GB + GlusterFS 3.6.9
Network	1Gb/s Ethernet

D. Model aggregation

Since not every stage of the analysis could be distributed, there is a step in which we aggregate a final matrix that will be used to compute an update matrix. This matrix is distributed afterwards, so 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 measure, and the following iteration starts.

6. Output persistence

After every iteration is executed, the output is stored to HDFS. This is executed in parallel, as every partition is stored independently.

4. EVALUATION

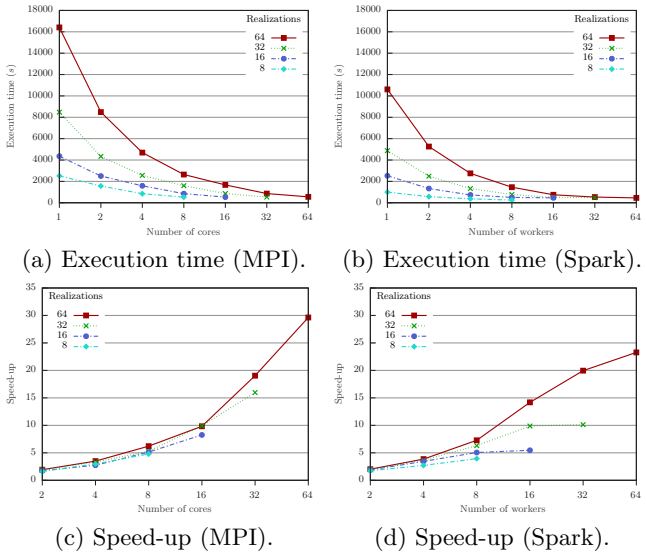
The goal of our evaluation is to assess the benefits and drawbacks of the application of the techniques discussed in Sec. 2. We focused on absolute execution time and speed-up to analyse the effects of the memory and virtualization overheads of Spark.

4.1 Experimental Setup

In order to assess the performance and scalability of the application, we selected a local cluster as baseline. The specifications and limitations of this testbed are described as follows.

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.

4.2 Execution Models: Spark vs. MPI



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

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

Figure 2: Execution time and speed-up for the MPI and Spark workflows, running on a local cluster.

The objective of these experiments is to detect the effects that 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 each Spark executor to one core in order to fairly compare scalability against single-core MPI processes. We run experiments increasing realization volumes and executors number. We measured the absolute execution time for a single execution (including the job launch time required by Spark) and computed the speed-up achieved. The results for those experiments are shown in Fig. 2, 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.

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. Consequently, the post-processing stage starts to affect the overall execution time more than with a fewer number of the parallel executors.

5. CONCLUSIONS AND FUTURE WORKS

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 these 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 to be compared: MPI and Apache Spark. We experimented with a specific workflow from the hydrology domain, and analyzed its performance and scalability implemented on MPI and Spark and executed in a local cluster.

The main conclusion from the experiments made 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.

While we considered this use case relevant for our objectives in terms of complexity and size, further experimentation with other workflows with different structures would be necessary to corroborate our results and conclusions. 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.

6. ACKNOWLEDGEMENTS

This work has been partially funded under the Spanish Ministry of Economics and Competitiveness grant TIN2013-41350-P, the COST Action IC1305 “Network for Sustainable Ultrascale Computing Platforms” (NESUS), and the FPU Training Program for Academic and Teaching Staff FPU15/00422 by the Spanish Ministry of Education. We gratefully acknowledge Wolfgang Kurtz (IBG-3, Forschungszentrum JÃijlich GmbH) for providing the EnKF-HGS simulator source code. We also acknowledge Oliver Schilling (CHYN, University of Neuchâtel) for providing the hydrological model use-case.

7. REFERENCES

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

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

[3] 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)

[4] Duro, F., GarcÃ■a, 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)

[5] 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)

[6] 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)

[7] 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)

[8] 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)

[9] 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), http://dx.doi.org/10.1007/s10723-015-9329-8

[10] 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)

[11] de Oliveira, D., Ogasawara, E., BaiÃčo, 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)

[12] 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)

[13] 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), http://dx.doi.org/10.1007/s10723-013-9282-3

[14] 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)

[15] 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)

[16] 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)

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

[18] 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)