Big Data and HPC collocation: Using HPC idle resources for Big Data Analytics

Michael Mercier, David Glesser, Yiannis Georgiou, Olivier Richard

Big Data and HPC collocation: Using HPC idle resources for Big Data Analytics

Michael Mercier*†, David Glesser†, Yiannis Georgiou†, Olivier Richard*,
*Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France
michael.mercier@inria.fr, olivier.richard@imag.fr
†Bull, Atos technologies
yiannis.georgiou@atos.net, david.glesser@atos.net

***Abstract*—Executing Big Data workloads upon High Performance Computing (HPC) infrastructures has become an attractive way to improve their performances. However, the collocation of HPC and Big Data workloads is not an easy task, mainly because of their core concepts' differences. This paper focuses on the challenges related to the scheduling of both Big Data and HPC workloads on the same computing platform.**

In classic HPC workloads, the rigidity of jobs tends to create holes in the schedule: we can use those idle resources as a dynamic pool for Big Data workloads. We propose a new idea based on Resource and Job Management System's (RJMS) configuration, that makes HPC and Big Data systems to communicate through a simple prolog/epilog mechanism. It leverages the built-in resilience of Big Data frameworks, while minimizing the disturbance on HPC workloads.

We present the first study of this approach, using the production RJMS middleware OAR and Hadoop YARN from the HPC and Big Data ecosystems respectively. Our new technique is evaluated with real experiments upon the Grid5000 platform. Our experiments validate our assumptions and show promising results. The system is capable of running an HPC workload with 70% cluster utilization, with a Big Data workload that fills the schedule holes to reach a full 100% utilization. We observe a penalty on the mean waiting time for HPC jobs of less than 17% and a Big Data effectiveness of more than 68% in average.

***Index Terms*—HPC and Big Data Convergence, Infrastructure, Scheduling, Best Effort, Resource Management.**

I. Introduction

The amount of data produced either in the scientific community or the commercial world, is constantly growing. This growth induces changes of scales on the processes of collecting, processing and storing data. The field of Big Data has emerged to handle these new challenges. The processing tools available in the Big Data community are easy to use but lack computation performances.

One of the main objective of High Performance Computing (HPC) field is to provide infrastructures to make computing as fast as possible, at the largest possible scale. To achieve performance, HPC users are using low level programming models that make applications hard to write and maintain.

Thus, Big Data and HPC convergence is inevitable. However, even though both fields share a common objective, they do not share the same core concepts. HPC infrastructure can be defined by a set of computing resources with low latency interconnect and a Parallel File System (PFS) that provide fast storage for all of these resources. HPC workloads consist in a set of rigid jobs with a strict resource and time requirement (i.e. 10 CPU for 2 hours). This time requirement is called walltime: it is not a prediction of the execution time but an upper bound after which the job will be killed.

On the other hand, a Big Data workload is made of malleable jobs that have requirements on multiple resources but no time requirements (i.e. 4 CPU and 10GB of memory). This workload is made of any type of task-oriented jobs that is able to cope with a dynamic resource allocation: Hadoop MapReduce stands in this category, but also more complex frameworks like Spark or Flink. It uses a Distributed File System (DFS) over all the compute nodes to store datasets, using replication to avoid data loss and load balance the data. It comes with the great advantage that most of the data movement can be avoided by bringing computation directly where the data is stored.

Our goal is to provide a way to collocate traditional HPC workloads and Big Data workloads on the same HPC infrastructure, in a way that one user can submit HPC and/or Big Data jobs directly to the RJMS, like he is used to do, without modifying the resource managers. We propose a new system called BeBiDa that performs Best effort Big Data Analytics on HPC infrastructure. To get the best of both world, we keep the PFS and the DFS leveraging their specialization for both workloads.

Collocation is interesting for owners, users and operators of these platforms:
From the **owners** point of view, collocation increases the platform utilization and attracts new users to HPC clusters by providing Big Data tools alongside traditional HPC tools;
From the **users** point of view, collocating HPC and Big Data workloads will permit new workflows that mix both approaches and benefit from the best computation tools from the HPC field and the best data analysis tools from the Big Data field;
For the **operators** point of view, HPC systems, on their own, are producing a large amount of data from hardware counters and services logs. The administrators of HPC systems could take advantage of Big Data tools to do constant analytics of these data in order to detect silent failures, user misuses, configuration errors and so on.

To reach our objective we don't want to provide yet another

tool that claims to manage every possible workload while reducing the number of specific features available. We want a solution as simple as possible without loosing features. Moreover, RJMS are massive industrial grade projects highly optimized for their specific workloads: for instance, Slurm (RJMS for HPC) and Yarn (RJMS for Big Data), have more than 45k effective line of code[1]. That is why we want to fully leverage the advantages of each middleware on their area of expertise.

The contributions of this paper are:

1) a definition of the HPC and Big Data collocation problem
2) a new method, called BeBiDa, for collocating HPC and Big Data workloads
3) an implementation of this method using industrial grade RJMS
4) an in-depth analysis of this implementation

The remainder of this paper is organized as follows: First, we define problems regarding the assumptions listed in Section II. Then, we describe the original idea of simple communication between RJMS with prolog/epilog to make them interact with different priority levels in Section IV. Section V provides a proof of concept implementation of our approach and the results of its evaluation. Finally, Section VI concludes this paper.

II. Definition of the Collocation Problem

We are interested in the problem of collocating a Big Data RJMS and an HPC RJMS, in order to run a Big Data workload ($W_{BigData}$) and an HPC workload (W_{HPC}) on the same cluster. Here is a general definition of this problem.

We defined M as the set of all the considered machines (a.k.a resources). The machine set M is divided into groups defined as follows:

- $M_{HPC-only}$ are dedicated resources for W_{HPC}, they are not visible for the Big Data RJMS
- $M_{BigData-only}$ are dedicated resource for $W_{BigData}$, they are not visible for the HPC RJMS
- M_{shared} is the part of the cluster that is shared between the two systems: it can be used to execute jobs from both $W_{BigData}$ and W_{HPC}.
- M is union of all these sets:

$$M = M_{HPC-only} \cup M_{shared} \cup M_{BigData-only}$$

Note that, if $M_{shared} = \emptyset$ we go back to a static partitioning of the cluster. On the other hand, if $M_{BigData-only} = \emptyset$ and $M_{HPC-only} = \emptyset$ then, the cluster is entirely shared.

Also, to be able to talk about HPC jobs and Big Data applications seamlessly, in the rest of this article, the generic term *jobs* will be used for both.

[1]Computed with cloc on the official GitHub repository excluding blank and comment lines

For the sake of clarification, the following list presents the assumptions that are used for the rest of this paper:

- The cluster contains a set of machines M
- One HPC RJMS is set up, and have access to M_{shared} to schedule jobs of from W_{HPC}.
- One Big Data RJMS is set up, and have access to M_{shared} to schedule jobs from $W_{BigData}$.
- All the software stack is managed by an administrator and not by the users. Both systems are already configured to start their jobs: no deployment needed.
- Jobs in W_{HPC} use a dedicated Parallel File System (PFS) to store data.
- Jobs in $W_{BigData}$ use a Distributed File System (DFS) that is spread over a set of machines:

$$M_{DFS} \in M_{BigData-only} \cup M_{shared}$$

- HPC Jobs are statically allocated (rigid or moldable).
- Big Data jobs are malleable and resilient to the interruption of all, or part of, their resource allocation.

III. Related works

At application level, the Big data software stack is being transformed to take advantages of the high performance hardware provided in HPC, like fast interconnect and fast Parallel File Systems (PFS) [19]. Some are taking the opposite direction, and use HPC traditional technologies, like MPI, to build new Big Data tools because the resilience of the Big Data software stack do not compensate for the lack of performances [16]. Thus, we conclude that it is worth to use the traditional Big Data tools on HPC infrastructure only if we leverage their dynamicity and resilience.

At resource management level, it exists multiple ways to make Big Data and HPC workloads run together.

The most obvious one is to have two different clusters which will be dedicated to each workload. But, data transfer between the two clusters will be a serious bottleneck. Moreover, there is no load balancing between the clusters because of the strict separation of concern.

Another simple approach is to let the user manage his own Big Data software stack inside HPC batch jobs using a set of scripts [5]. It works for simple workflows with a small amounts of data but it have drawbacks: (a) Even if the scripts can help, the user has to operate and configure part of the system by himself and it can be complex while the software stack grows. (b) The Big Data software stack has to be deployed, configured, started and shutdown for each job: the overhead is important, especially for small jobs. (c) If the amount of data is too big to fit in the user home, a third-party storage would be needed and the data mouvement produced by data ingestion can dominate the execution time [11].

One more idea would be to run HPC jobs within a Big Data stack. The bigger problem is that most HPC applications are not able to run within Big Data RJMS because they need to implement a communication layer with it. A rewrite of most HPC applications is required which makes this approach unfeasible in the HPC community.

An integrated approach is to add a new abstraction level to integrate one system to the other with an adaptor that convert Big Data RJMS resource requirements into HPC RJMS allocations [1], [15]. But these approaches are tightly coupled to specific technologies, have to evolve with them and are limited to them. It restricts this solution to the implemented adaptors, and the cost of maintaining this whole new layer can be too high.
In comparison, our approach is based on configuration and the only code that has to be adapted while changing technologies is the prolog/epilog scripts that count 60 lines of bash in our implementation.

A more sophisticated approach is to encapsulate HPC on Hadoop or, Hadoop on HPC with a Pilot-based abstraction [12]. The authors provide a new layer of software that glue the two systems.
The main reason we take another approach is the complexity of this solution. Also, it relies heavily on Hadoop while our approach is more technology agnostic and capable to adapt to any kind of system that is able to dynamically commission and decommission resources, and any HPC scheduler that have a prolog/epilog mechanism.

One of the key aspects of Big Data on HPC is the data management. We made the choice to use a DFS to avoid the traditional HPC IO bottleneck and scale linearly in terms of bandwidth [20], without adding expensive hardware to scale up the PFS [8].

More than a decade ago, when desktop grids and volunteer computing have been introduced, systems such as Boinc [2] and Ourgrid [3] have started handling collocation of different types of workloads using similar techniques like ours. In the same area, the lightweight grid meta-scheduler Cigri [9] is based on OAR's RJMS best-effort jobs type in order to schedule bag-of-tasks grid workloads when the local HPC jobs leave idle resources on the cluster. BeBiDa mechanism is based on the same idea of exploiting idle resources but without introducing another level of scheduling.

To the best of our knowledge, the approach presented in this paper is the first to use two unmodified RJMS that collocate HPC and Big Data workloads on the same cluster, without adding a third tool to coordinate them.

IV. BeBiDa solution description

A. Description

The main idea of BeBiDa is to run two industrial grade RJMS collocated on the same cluster: One Big Data RJMS and one HPC RJMS. Big Data is running in best-effort, i.e., it uses **HPC idle resources** as a **dynamic resource pool**.
It relies on the prolog and epilog mechanisms that are provided by most HPC RJMS: The prolog (resp. epilog) is a script that is executed before (resp. after) the execution of each job.
Here is detail description of this process:

1) By default, all resources are attached to the Big Data resource pool
2) when an HPC job starts: the prolog decommissions the resources needed by the HPC job from this pool
3) When an HPC job finishes: The epilog is giving the resources back to the Big Data resource pool

From our solution derives the following additional assumptions:

- W_{HPC} as priority over $W_{BigData}$
- The set of collocated machines M_{shared} are seen by the HPC RJMS as always available.
- The Big Data RJMS sees only the part of M_{shared} that is not running HPC job as available. This part is dynamic over time.

B. Advantages and drawbacks

The main advantages of this solution is to be easy to implement and as simple as a configuration problem for both RJMS: it do not requires any third tool, it avoids tight coupling between the two systems and can be easily implemented against most of the HPC and Big Data RJMS.

This configuration have to be done by the cluster administrator that manage both RJMS. The solution is transparent to final users: they submit HPC and Big Data jobs thought there respective traditional interfaces.

Only small scripts are needed (60 lines of bash) for HPC RJMS prolog and epilog, no code modification is needed for any tool inside both software stack.

Also, it leverages Big Data frameworks resilience and dynamicity by using a dynamic resource pool and do not disturb the HPC applications by ensuring that no processes left on the compute nodes after decommissioning, except for the DFS daemon. The impact of this daemon is beyond the scope of this publication.

It also impact HPC jobs' waiting time due to the prolog/epilog execution time: this is discuss in the V-B section.

For the Big Data workload, resource preemption during a job execution can be a problem: if it happens too often, its cost can be unsustainable and Big Data computation will become unpredictable. This is also measured in the V-B section.

Obviously, this approach will perform well when Big Data is dominant in the workload; i.e. when the HPC utilization is low. On the opposite, we expect to have poor Big Data performance on a loaded HPC cluster because the number of killed -on duty- workers will increase, inducing more re-computation and data movement.

V. Experimentations with BeBiDa

Our goal in this section is to describe our BebiDa proof of concept implementation, and to present the experiment's design and results.

A. BeBiDa implementation

In this implementation, no modifications have been done to the RJMS. Only their configuration have been adapted to fit our needs. Configuring industrial grade software can be seen as a straightforward task but it is not. Both RJMS have hundreds of parameters that needed to be set (scheduling,

resources, accounts, security policies, file systems, ...). All code and experiment scripts can be found in a public Git repository[2]. These experiments are implemented using State-of-the-Art tools for reproducibility [17] [10].

1) Hardware: Our experiments run on the Grid5000 infrastructure [4]. We choose the Graphene cluster on the site of Nancy: It has 1x Intel Xeon X3440 with 4 cores/CPU, 16GB RAM and 298GB HDD. We select $M = 34$ nodes, with $M_{shared} = 32$ and $M_{BigData-only} = 2$ and an additional master node to host YARN and OAR masters.

2) Software: We use OAR 2.5.4 as HPC RJMS. OAR is easy to deploy and configure and integrates well with our testbed.

Prolog and epilog scripts are added as hooks that are executed before and after each HPC jobs on OAR's configuration.

We choose YARN from Apache Hadoop 2.7.3 as the Big Data RJMS. With Yarn, we use Spark as computation framework and HDFS as distributed file system.

Spark 2.1.0 is configured to run 2 executors by node and all the applications are run in client mode with 1 core and 1GB of memory for the application master.

Also, because Spark Application Master is a single point of failure for the application, we use the node label capabilities of the YARN capacity scheduler and of Spark to pin the Application master to the $M_{BigData-only}$ nodes. As side effect, this setting limits the number of concurrent running applications to the number of Application Master $M_{BigData-only}$ can host.

3) Workloads: For this experiment we need mixed workloads of HPC and Big Data jobs but, because such workloads are pretty new, we do not have access to real traces for now. To be able to test our system on different HPC and Big Data workloads we chose to generate those workloads based on the statistical study done in [7]. But, to generate these workloads we need job profiles for each type of workloads.

The Big Data workloads are composed of 3 different type of applications taken from the BigDataBench benchmarks [18]: Grep, WordCount and K-means. We choose to use 3 size of datasets for each application (32GB, 64GB, and 128GB) that are generated using the BigDataBench generation tool [13]. These datasets are injected into HDFS at the beginning of the experiment with a replication factor of 3.

The HPC workloads are composed of 3 different applications type taken from the NAS Parallel Benchmarks [14]: IS, FT and LU. Those applications are compiled for different number of resources, each power of two from 2^2 to 2^7 (the number of cores in the cluster), and different size from C to E. Then these job profiles are filtered to obtain a 14 jobs profiles that run for at least 1 minute and less than half an hour. The workloads generated contains 250 jobs randomly picked in this pool using previously mentioned method.

We use our simulator, Batsim [6], to select the generation parameters in order to get an average utilization around 70%. We have selected this level of utilization because it is representative of small to medium size HPC center.

[2]https://gitlab.inria.fr/mmercier/bebida

B. Results

With this experiment, we want to answer the following questions:

1) Does BeBiDa work in real conditions?
2) What is the overall utilization of the cluster?
3) Is the Big Data computation effective in these conditions?
4) What is the overhead on the HPC workload?

First, the system works correctly with the aforementioned configuration and setup: It is able to run HPC jobs normally and Big Data jobs in the holes of the HPC scheduling. You can see Gantt chart of one experiment instance in Figure 1. To show this level of details we have chosen one experiment instance. We try to pick a representative experiment regarding HPC utilization. Note that in the Big Data workload each band represented a Spark executor with a common color for each application. Spark is configured to have 2 executors per node and when this is the case the executors are overlapping and the colors are mixed. You can also notice that, as the prolog/epilog scripts are very simple, they do not have a vision of what will happen in the future: that's why even if two HPC jobs are consecutive executors they are started during the guard time of 60s that OAR puts between 2 jobs. Also, we can notice that executors are overlapping on HPC jobs: this is due to the fact that Spark induces a delay between the real kill of an executor and the report of this kill in the application logs.

Having a full cluster utilization is one of the HPC owner main goals. With BeBiDa, The overall utilization of the cluster virtually goes from 70% on average to 100% if the $W_{BigData}$ contains enough work to fill the holes left in the schedule by W_{HPC}. But, this statement is mitigated by 2 points: First there is a small delay when resource goes back to idle state before it is actually used to compute $W_{BigData}$ which correspond to the Big Data scheduler's delay; Second, resource preemption cost on the Big Data jobs reduces their work efficiency. The cost of this preemption highly depends on the computation framework capacity to manage these events. To tackle this last point we define the efficiency E of the computation done by the Big Data framework, here Spark. It can be computed with the time spent in tasks that are complete and not resubmitted over the total task computation time:

$$E = \frac{\sum T_{complete} - T_{resubmitted}}{\sum T_{complete} + T_{failed}}$$

To understand this metric the reader need to understand how Spark works: It uses an internal in-memory representation called RDD [21] that uses lazy evaluation and lineage to recompute only what is necessary if some intermediate data is missing. When a preemption happens, currently running tasks are killed; computation time is lost (T_{failed}), and these results are lost but also the intermediate data of previously successful tasks have to be recomputed elsewhere: this is the time resubmitted tasks ($T_{resubmitted}$). Figure 2 shows the effectiveness goes from 44% to 91% with a mean of 68%.

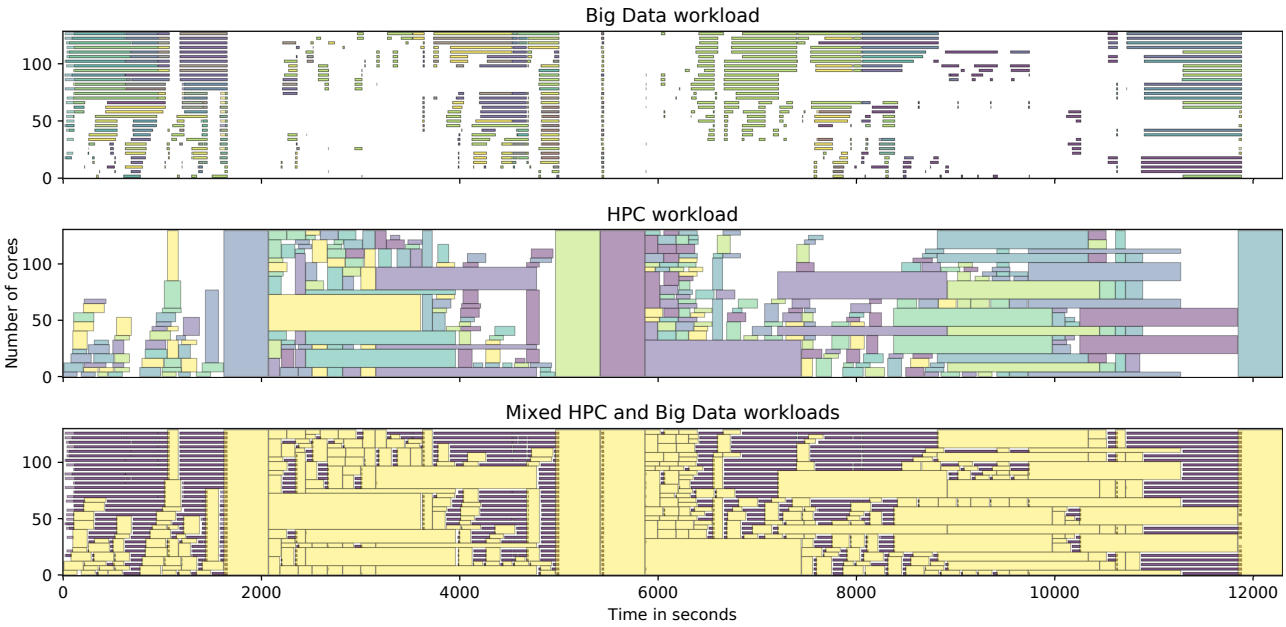

Fig. 1. Example of one experiment with BeBiDa enabled

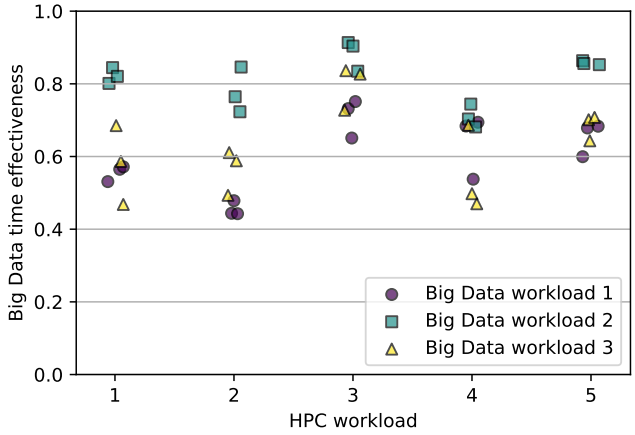

Fig. 2. $W_{BigData}$ Time effectiveness E regarding HPC workload utilization

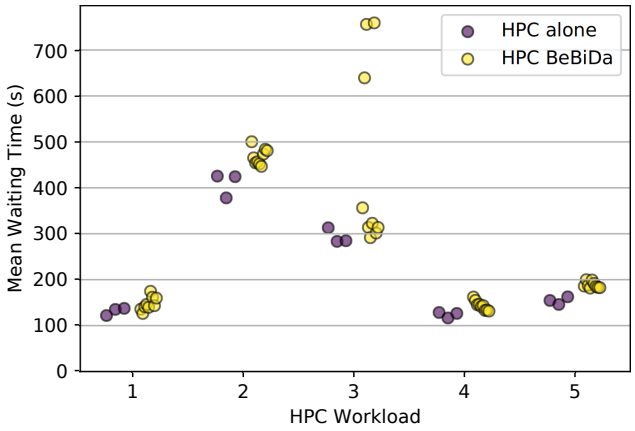

Fig. 3. Overhead on W_{HPC}

We also measure the time taken by prolog and epilog themselves.

The prolog hook happen when when an HPC job is allocated: it decommissions the resources associated to this job by killing YARN containers if present, and the NodeManager afterwards on these resources. This process (i.e. the prolog) takes from 2 to 16 seconds depending if there is running containers or not: It takes around 16s if there is 1 or multiple containers.

The epilog process that restart the NodeManager takes less than 3 seconds.

Both hooks implies a small penalty on each job's waiting time that can seen in Figure 3. But, The mean waiting time overhead is less then 17% on average.

We can notice that, for some measures, the workload 3 mean waiting time's is highly impacted: this is due to a bad backfilling decision on a big job that delayed a lot of small and medium jobs. It only happens in specific condition that are not met at every run. It illustrates the highly sensitive nature of scheduling where small changes can dramatically impact the whole system.

All experiment combination are executed 3 times.

The mean execution time of HPC jobs is also impacted by BeBiDa: It is increased by 6% in average. This is due to the network contention created by $W_{BigData}$ and the memory and

computation overhead due to the HDFS daemon that is still running during HPC job execution.

VI. Conclusion

This paper defines the problem of collocating HPC and Big Data workloads on the same HPC cluster. We propose a new approach called BeBiDa that is seamless for end users and requires only configuration from the cluster administrator. It is based on a simple job prolog/epilog mechanism, which is very common on HPC batch schedulers. This solution exposes all nodes that do not run HPC jobs to the Big Data resource manager as a dynamic pool of resources.

We provide a proof of concept and run experiments over it. It shows that with an HPC utilisation of around 70%, the system is able to run Big data jobs on the unused resources. The high dynamicity of the Big Data resources pool induces a loss of efficiency. We define the time effectiveness metric to mesure this efficiency. In our experiments, the time effectiveness goes from 44% to 91% (with a mean of 68%) depending on the Big Data and the HPC workloads. As a future work, we will implement a model of this system over the batsim simulator to be able to explore a wider range of parameters.

We also want to study the interferences of a DFS on HPC applications is order to limit its impact. Our approach can be extended to any systems with the notion of scheduling with priority. Finally, finding good metrics that can be optimized in new scheduling heuristics remains a big challenge.

VII. Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). We also want to thank the KerData team from Inria Rennes for their support and their interest in our work.

References

[1] intel-hpdd/scheduling-connector-for-hadoop: HPC Adapter for Mapreduce/Yarn(HAM).

[2] D. P. Anderson. Boinc: a system for public-resource computing and storage. In *Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Nov 2004.

[3] Nazareno Andrade, Walfredo Cirne, Francisco Vilar Brasileiro, and Paulo Roisenberg. Ourgrid: An approach to easily assemble grids with equitable resource sharing. In *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*, pages 61–86, 2003.

[4] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In IvanI. Ivanov, Marten Sinderen, Frank Leymann, and Tony Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[5] Albert Chu. chu11/magpie, October 2013.

[6] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator. In *20th Workshop on Job Scheduling Strategies for Parallel Processing*, Chicago, United States, May 2016.

[7] Dror G Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, Cambridge, 2015.

[8] Niall Gaffney, Christopher Jordan, Tommy Minyard, and Dan Stanzione. Building wrangler: A transformational data intensive resource for the open science community. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 20–22. IEEE, 2014.

[9] Yiannis Georgiou, Olivier Richard, and Nicolas Capit. Evaluations of the lightweight grid CIGRI upon the grid5000 platform. In *Third International Conference on e-Science and Grid Computing, e-Science 2007, 10-13 December 2007, Bangalore, India*, pages 279–286, 2007.

[10] Matthieu Imbert, Laurent Pouilloux, Jonathan Rouzaud-Cornabas, Adrien Lèbre, and Takahiro Hirofuchi. Using the EXECO toolbox to perform automatic and reproducible cloud experiments. In *1st International Workshop on UsiNg and building ClOud Testbeds (UNICO, collocated with IEEE CloudCom 2013*, December 2013.

[11] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego*, 2011.

[12] Andre Luckow, Ioannis Paraskevakos, George Chantzialexiou, and Shantenu Jha. Hadoop on HPC: Integrating Hadoop and Pilot-based Dynamic Resource Management. *arXiv preprint arXiv:1602.00345*, 2016.

[13] Zijian Ming, Chunjie Luo, Wanling Gao, Rui Han, Qiang Yang, Lei Wang, and Jianfeng Zhan. BDGS: A scalable big data generator suite in big data benchmarking. In *Workshop on Big Data Benchmarks*, pages 138–154. Springer, 2013.

[14] NASA. NAS Parallel Benchmarks, February 2016.

[15] Marcelo Veiga Neves, Tiago Ferreto, and César De Rose. Scheduling MapReduce Jobs in HPC Clusters. In Christos Kaklamanis, Theodore Papatheodorou, and Paul G. Spirakis, editors, *Euro-Par 2012 Parallel Processing*, number 7484 in Lecture Notes in Computer Science, pages 179–190. Springer Berlin Heidelberg, August 2012. DOI: 10.1007/978-3-642-32820-6_19.

[16] Jorge L. Reyes-Ortiz, Luca Oneto, and Davide Anguita. Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *Procedia Computer Science*, 53:121–130, January 2015.

[17] Cristian Ruiz, Salem Harrache, Michael Mercier, and Olivier Richard. Reconstructable software appliances with kameleon. *SIGOPS Oper. Syst. Rev.*, 49(1):80–89, January 2015.

[18] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, and others. Bigdatabench: A big data benchmark suite from internet services. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 488–499. IEEE, 2014.

[19] M. Wasi-ur Rahman, Xiaoyi Lu, N.S. Islam, R. Rajachandrasekar, and D.K. Panda. High-Performance Design of YARN MapReduce on Modern HPC Clusters with Lustre and RDMA. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 291–300, May 2015.

[20] Pengfei Xuan, Jeffrey Denton, Pradip K. Srimani, Rong Ge, and Feng Luo. Big Data Analytics on Traditional HPC Infrastructure Using Two-level Storage. In *Proceedings of the 2015 International Workshop on Data-Intensive Scalable Computing Systems*, DISCS '15, pages 4:1–4:8, New York, NY, USA, 2015. ACM.

[21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.