# On Many-Task Big Data Processing: from GPUs to Clouds

Ana Lucia Varbanescu
University of Amsterdam
The Netherlands
A.L.Varbanescu@uva.nl

Alexandru Iosup
Delft University of Technology
The Netherlands
A.Iosup@tudelft.nl

## ABSTRACT

Big data processing relies today on complex middleware stacks, comprised of high-level languages, programming models, execution engines, and storage engines. Among these, the execution engine is often built in-house, is tightly integrated or even monolithic, and makes coarse assumptions about the hardware. This limits how the future big data processing systems will be able to respond to technological advances and changes in user workloads. To advance the state-of-the-art, we propose a vision for an open big data execution engine. We identify four key requirements and design a modular, task-based architecture to address them. First, to address performance requirements, we envision the use of multi-layer parallelism, which enables the efficient use of heterogeneous platforms (from GPUs to clouds). Second, to address elasticity, we propose the use of online provisioning and allocation of cloud-based resources, which enables economically feasible processing of many big data tasks. Third, to address predictability, we define a performance envelope, which enables early decisions for selecting and sizing execution platforms. Fourth, we characterize the interaction between our execution engine architecture and the other layers, to enable the efficient operation of the entire big data processing stack. We further identify and analyze eleven challenges in realizing our vision. Ultimately, these challenges will inspire and motivate a new generation of execution engines for big data processing.

## Keywords

Large scale big data processing, Execution engine, Predictability, Elasticity, High-performance

## 1. INTRODUCTION

Driven by the steadily increasing volumes of data from many fields of science and society, ranging from biology to astronomy, and from social networks to finances, big data processing is one of the immediate challenges of computing. Complex analysis of big data is of interest for many companies and fields, being an important source of both (increased) revenue and scientific discovery [1, 2]. Timely analysis is as important, making big data applications a new field of high-performance computing (HPC) [3]. Although many big data processing stacks already exist, they are monolithic, highly integrated stacks. This lack of flexibility makes the reaction time to changes in workloads and/or platforms unacceptably high, especially for common users. To tackle this problem, this work describes our vision for a big data processing stack that is not monolithic.

Many of the companies that drive the data analytics business are small and medium enterprises (SMEs), which focus mainly on the design and development of new models and analysis tools. They typically[1] process average-sized datasets—100s of GBs to TBs of data—but cover a broader range of algorithms and domains [5,6]. This dynamic mix of data and computation makes it difficult to estimate the number and type of permanent resources to cover the request, making dedicated supercomputers and/or clusters unfeasible, both time- and money-wise.

In contrast to dedicated infrastructure, on-demand access to compute resources is much more attractive for these dynamic enterprises, allowing control over the type and number of resources to be used [7]. Cloud computing has emerged in the past few years as an IT paradigm in which the infrastructure, the platform, and even the software used by IT operations are outsourced services; Infrastructure as a Service (IaaS) clouds can lease compute and storage resources to their users. Services can be used flexibly (i.e., when and only for how long they are needed), and leased for prices charged in small increments according to the actual usage. Amazon, Microsoft, Google, and SalesForce are only four examples out of more than one hundred commercial cloud providers.

Commercial IaaS clouds offer a wide diversity of (virtual-

[1]It is symptomatic for this developing field that the actual types of data and the classes of algorithms used in practice can only be guessed at this moment. Our recent survey [4] of over 100 articles published in the field reveals rather small datasets, with maximum size of only a few tens of GBs, and only few classes of algorithms, perhaps less than 10, being used in experimental work published mostly by *academia*. We are currently conducting a survey, across *academia and industry*, to find out about big data processes in practice: `https://docs.google.com/forms/d/132oJgFx7rZdZ_9aWxGI_LUq9hI1erqB9WqGVeEFi2_8/viewform`.

ized) resources, including CPUs of multiple generations and number of cores, GPUs, and even more exotic architectures. Combining these resources leads to an unprecedented mix of parallelism (at multiple-layers) and heterogeneity, which need to be addressed at both the application and the execution engine level [8, 9]. Moreover, systems that lease resources from clouds hold the promise of efficiency by being elastic, that is, by varying in scale over time. Therefore, for big data processing systems to achieve good performance, addressing multiple existing layers of parallelism (i.e., from very fine to very coarse-grained) and dynamic scaling (i.e., on-demand increase of resources) are essential.

Currently, big data processing relies on users implementing their applications in a high level programming model, making implicit choices about the execution model to be used, often hindering efficient resource usage. For example, when using a MapReduce-like model, users implicitly assume a specific way of dividing the application into homogeneous tasks of predefined sizes, which are then mapped, scheduled, and executed according to a programming model policy. System elasticity or hardware heterogeneity and multi-grain parallelism cannot be taken into account easily: instead, a new programming model would have to be chosen, and the development process restarted.

Achieving high performance is an important goal for SMEs. Although SMEs may also value cost and other non-functional attributes of their applications, these attributes may often be seen as a trade-off in relationship with performance. For example, an SME may aim to obtain the result of data processing within a deadline, for a certain cost limit; this is essentially a problem of high performance computing when the deadlines are tight (e.g., when working for a contractor, as many SMEs do) and when the cost model is simple (e.g., the per-hour charge of many commercial IaaS clouds). An another example, an SME may strive for a measure of efficiency expressed as the maximum amount of processing for a given budget; again, this problem requires achieving high performance.

We argue that, for SMEs, meeting the high performance requirements of a large diversity of big data workloads can be efficient only when the heterogeneity and layered parallelism of modern computational systems (from GPUs to clouds), is leveraged. Instead of addressing these quick-paced changes in the resources mix we are currently using for each programming/execution model, we envision a novel execution engine that allows for a flexible match between the application and the computational resources, without being directly bound to a programming model. In this article, we focus on the design of this generic execution model, emphasizing its design requirements and restrictions, and discussing the main challenges to be addressed for its performance and compatibility with the rest of the big data processing stack. Towards our vision, we identify and expose the potential bottlenecks in the design and prototype process of such a generic engine, allowing the community to contribute to their solving.

Our main contribution is therefore threefold: (1) We propose a vision for an open, non-monolithic big data processing architecture, suitable for SMEs. (2) We identify the key requirements for designing such an architecture. (3) We enumerate the first key challenges (below fifteen of them) that can and should be solved for this vision to become (closer to) reality.

The remainder of this article is structured as follows. Section 2 introduces the state-of-the-art in big data processing stacks. Section 3 presents the design requirements and the architecture of our execution engine, while Section 4 lists the most important challenges to be solved when implementing it. Section 5 concludes the article with a summary of our findings and contributions.

# 2. BACKGROUND

In this section we introduce the general architecture of a big data processing system and discuss the state-of-the-art, for which we identify high-level problems.

## 2.1 Big Data Processing Stack

Big data processing applications make use of increasingly more complex ecosystems of middleware. Figure 1 depicts the big data ecosystem, where systems-of-systems are all similarly structured into a four-layer big-data processing stack. This ecosystem is based on a variety of open-source middleware, both commercial and freeware, of increasing maturity but with widely different features and performance characteristics. We discuss each of the four layers, in turn.

The *High-Level Language* allows non-expert programmers to express complex queries and general data-processing programs in a data manipulation language. A variety of languages exist, many derived from the SQL92 standard (e.g., Pig Latin [10, 11]) or domain-specific (e.g., Spark [12] for multi-pass algorithms, especially common in machine learning). These languages are supported in the big data ecosystem by compiling tools that convert from these languages to specific programming models.

The *Programming Model* is currently the central layer of the big data stack. Although a variety of programming models exist, several, for example MapReduce, have started to become more commonly used. Similarly to the case of general programming languages, it is common for the choice for a particular programming model to be made based on perceived popularity, rather than features and proven performance; for example, the use of the MapReduce programming model for graph processing has been shown to lead to very poor performance in practice, when compared with other data processing stacks, yet it is still widely used. Tools can compile programs expressed in the programming model to programs that can be executed on the available resources by an execution engine.

The *Execution Engine* provides the automatic, reliable, efficient use of computational resources to execute the jobs derived from programs expressed in a particular programming model; although more general execution engines may appear soon, the current generation offers native support only for one programming model. Numerous execution engines exist; for the MapReduce programming model, Hadoop and its newer incarnation YARN are open-source, free-to-use execution engines. Depending on the characteristics and quality-of-service requirements of the application, the execution engine may require close interaction with a storage engine.

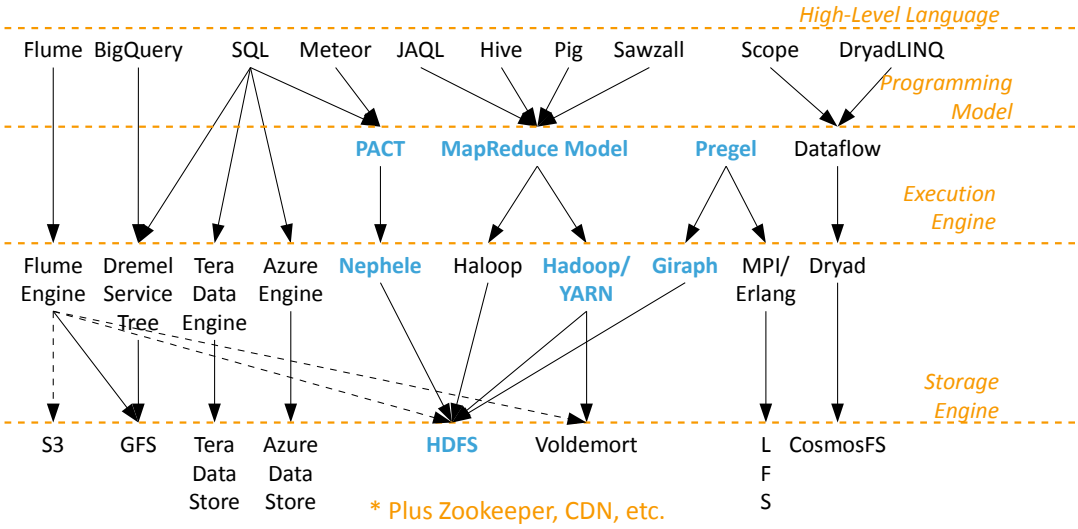


**Figure 1: A view into the ecosystem of Big Data processing.**

The *Storage Engine* provides hardware/software storage for big data applications. It typically provides a common file-system or key-value interface for storing data. The Hadoop Distributed File System (HDFS [13]) is a commonly used storage engine in the MapReduce stack.

These fours layers are not the only ones appearing in a big data processing stack. Necessarily, a *resource substrate* provides the actual resources (physical or virtualized). Another layer contains very diverse middleware, to provide state management services (e.g., Zookeeper), content management, etc.

## 2.2 Problem: The Monolithic Approach

Despite the high diversity in the big data ecosystem, the state-of-the-art in stack deployments is based on *monolithic*, highly integrated stacks. Current programming models, which may be the first to be fixed in a decision to install a new data processing unit, include and implicitly hide all the details regarding execution. Annotations could help, but a diverse set of useful annotations and a comprehensive analysis of the usefulness of existing annotations do not currently exist. The execution engine is typically deployed on a fixed set of resources, can sometimes execute no more than a few big data processing jobs concurrently, and can rarely coexist with other execution engines in the same environment. The storage engine exhibits similar issues to the execution engine, although for large volumes data, instead of for relatively small volumes of data and large amounts of computation per deployed compute unit.

The monolithic approach has democratized big data processing, but at the expense of significant waste of compute and storage capability. Understanding the performance of a big data application, from the information exposed to the programmer, is severely hindered by the aspects abstracted away, such as the mapping and scheduling of applications to resources, provisioning of needed resources (and their release), and the actual execution of the application.

To address these high-level problems that are inherent in current big data stacks, we propose in the following section a generic architecture that promises to address them at the level of the execution engine. Our focus on the execution engine is motivated by the variety of resources managed at this level, including in particular heterogeneous compute elements.

# 3. A GENERIC ARCHITECTURE FOR MANY-TASKS BIG DATA PROCESSING

In this section we discuss the requirements and the design of a generic execution engine for a *many-tasks* big data processing.

## 3.1 Requirements

We envision big data processing stacks that exploit affordable computational capacity, flexible in scale and composition. Such a set of machines, either leased for short periods of time from a commercial IaaS cloud or even common computers clustered together, is likely to be equipped with multi-core processors and GPU-like accelerators of different capabilities. Therefore, heterogeneity and a large variety of parallelism and performance capability must be expected.

For such computational infrastructures, much poorer in communication capacity and latency than high-end dedicated supercomputers, intensive data communication between computation tasks will greatly reduce performance. To avoid this bottleneck, data processing applications should be partitioned into bags of many loosely-coupled or even independent tasks. Providing these tasks to the execution engine allows it to take into account both the available resources and the performance requirements to provide an efficient, high-performance execution of the workload.

To design a generic architecture for a performance-aware execution engine with access to different kinds of computational resources, we identify four driving requirements:

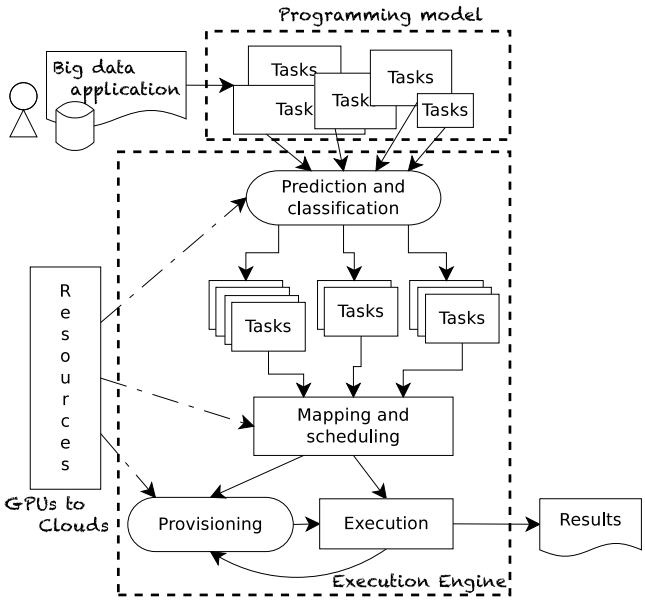


**Figure 2: A generic architecture for many-task big-data applications.**

1. *High Performance*: the robust exploitation of the inherent multi-layered parallelism of today's heterogeneous computational infrastructures must be leveraged (e.g., [9]). This requirement also enables higher-level trade-offs that may appear in the operation of an SME, such as performance-cost or performance-accuracy.

2. *Elasticity*: the elastic (up- and down-scaling) yet efficient provisioning and allocation of resources for a variety of workloads (e.g., [14]) must be enabled.

3. *Predictability*: the performance envelope of a given computational capacity, subject to different data sets, algorithms, and processing stacks, e.g., [4], should be estimated.

4. *Compatibility*: the integration of the execution engine with the high level programming model and the storage system must be provided.

## 3.2 Architecture

To address the four major requirements formulated earlier, we have designed a generic big-data processing architecture depicted in Figure 2. Responding to requirements, our architecture assumes that data-processing applications can fulfill the many-task paradigm: the architecture is designed to execute many tasks with high throughput.

We will further focus on the Execution Engine, built with four distinctive features. Firstly, this architecture decouples the execution engine from the input constraints provided by the programming model (that is, tasks) and by the resource substrate (that is, actual compute resources). Secondly, by clearly exposing to the user, from the Execution Engine, a set of components and their interconnections, this architecture enables the design, tuning, and implementation of each component, either in isolation or as an inter-operated system. Thirdly, by considering both the tasks and the resources in the "Prediction and classification" component, this architecture makes provisions for advanced prediction and classification mechanisms, and in particular for mechanisms supporting true resource heterogeneity (that is, from GPUs to clouds). Because applications can behave very differently on CPUs and GPUs, adequate mechanisms can offer important opportunities for improvement. By classifying tasks, this component also makes provisions for self-tuning mechanisms, relieving the naïve user from one of the burdensome tasks but without necessarily sacrificing performance. Fourthly, the provisioning mechanism can extend the functionality of popular Execution Engines, such as Hadoop's, with the ability to use an elastic set of resources; for example, it could enable the resource substrate to increase with resources leased from commercial IaaS clouds only when needed and for as long needed. These *transient resources* [15] offer new opportunities for efficient execution mechanisms.

Our architecture addresses the requirements as follows:

- *High Performance* requirements (addressed in Section 4.2) are fulfilled through the individual behavior and through the interplay of several components, primarily Mapping and Scheduling, Provisioning, and Execution.
- *Elasticity* requirements (Section 4.2) are fulfilled through the individual behavior and through the interplay of Provisioning and Execution.
- *Predictability* requirements (Section 4.3) are fulfilled through the individual behavior of Prediction and Classification, which takes decisions based on the available resources and tasks to classify the tasks in terms of resources.
- *Compatibility* requirements (Section 4.4) are addressed by the interfaces offered by the Execution Engine: the back-end of the programming model and the resource specification.

# 4. OPEN CHALLENGES

In this section we discuss an open list of challenges that come with the architecture proposed in Section 3. We see each of this challenges as surmountable in the next decade.

## 4.1 Performance challenges

### *4.1.1 Parallel architectures and algorithms*

Although a predominant consumer of computational resources, big data processing is still in its early phases when it comes to algorithms. Application-driven, the field proposes many new algorithms, many currently as early prototypes. This empirical, trial-and-error approach leads to a large sequential code base, still increasing at fast pace. With the high parallelism required by all novel HPC architectures, this code base currently goes through a process of systematic and efficient, yet also cumbersome and application-specific parallelization.

We argue that this a-priori process of sequential algorithm development is redundant. As all big data processing is to be done on parallel architectures, we believe the community should focus instead on designing and implementing novel

and efficient *parallel algorithms*, from the beginning. Effort should be put in classifying and characterizing big data applications [16, 17], simplifying paralleization and performance analysis. Moreover, analysis and prediction models are needed to understand the performance potential these algorithms show for each class of parallel architectures. These models should be designed, tuned, and validated early in the algorithm development stages, when feedback and changes are easier to address.

The process of designing and implementing new algorithms is also facilitated by the advances in compiler technology, but there are caveats. First, although techniques such as autovectorization have improved significantly in the last few years, they may still be limited to only a few application classes. Second, parallelizing compilers tend to be very conservative with the transformations they apply, and therefore very sensitive with respect to the input code. To tackle this sensitivity, programmers may end up developing solutions to trigger the right compiler behavior, thus targeting the compiler's idiosyncrasies rather then implementing efficient parallel solutions for the required problem. Third, compilers typically focus on improving the performance of the computation, while big data processing applications should probably focus more on addressing the irregular memory access patterns in large data structures. Overall, although compilers will continue to improve towards simplifying some of the tasks needed for application parallelization, we believe that the high demand for parallelism and performance requires approaches with higher throughput. We argue that the most efficient such approach must still focus first on building new algorithms for massive parallel and distributed processing on big data.

#### 4.1.2 Heterogeneous platforms

Using the developments of Top500 [2] in the last five years as trend forecasters, heterogeneous, massive parallelism for commodity (and cloud) machines is likely to continue, given that many types of applications are able to prove them effective for data-intensive, irregular applications [9, 18]. To use these platforms effectively, programmers need to address two important issues: *application and data decomposition*. We argue that solving these problems is essential for increasing platform utilization and thus leveraging more performance [9].

We note that this approach is facilitated by the many task programming model we have adopted, but still requires analysis and tools for mapping and scheduling suitable tasks to suitable processes. Using the results of the performance evaluation/prediction, these tools can improve overall efficiency, ultimately improving overall workload performance.

#### 4.1.3 Programmablity by portability

The heterogeneous and dynamic nature of the computational infrastructure we are considering poses significant programmability challenges: tasks have to be able to run efficiently on different types of archtiectures, from distributed nodes to many-core CPUs and GPUs. Ideally, they should also be ready to address new, exotic architectures that might be incorporated in the near future (e.g., fused processors or domain-specific processors).

[2]http://www.top500.org

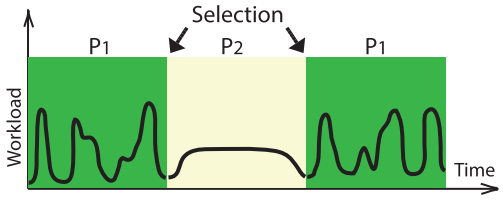


**Figure 3: Selected policy over the lifetime of a system with changing workload modes.**

Developing all these different versions of software is a huge programmability challenge. As new workloads are being designed and developed for new types of big data analysis, this challenge needs to be addressed in a systematic way. Therefore, defining and using portable programming models is the key to solve this challenge efficiently. Portable versions of all tasks are necessary to address both the elasticity and heterogeneity of the computational infrastructure. Implementing them in languages like OpenCL or OpenACC will provide a first level of functional portability, but more research is required to achieve acceptable levels of performance portability.

### 4.2 Elasticity challenges

#### 4.2.1 Performance and cost awareness under elasticity

In an elastic system, provisioning and allocation mechanisms should ensure that system can elastically increase its set of resources when overloaded and elastically decrease them when underloaded. Elastic operations should not impact the performance observed by or the cost incurred on system users—performance and cost awareness, respectively. For example, elastic MapReduce systems should not lose performance when resizing, in particular when sharing resources with other users. Recent work (i.e., [15] and references within) has proposed designs to facilitate the on-demand deployment of elastic MapReduce clusters in multi-cluster systems. However, extending these designs for general data processing is non-trivial: programming models such as dataflow (e.g., graph-processing Pregel, as implemented by Giraph) have different task coupling and resource usage patterns, possibly leading to different optimal performance and cost awareness approaches.

#### 4.2.2 Portfolio Scheduling

The schedulers of data processing systems may under-perform when the operational characteristics for which they were designed change: under unexpected workload arrivals, evolving workload characteristics of single or multiple users, new platform architectures, etc. Developing new scheduling policies and mechanisms to address the change involves various risks

and may only provide a temporary solution. To alleviate this problem, we have introduced in previous work [19] [20] [21] a general scheduling approach for data center environments, *portfolio scheduling*, under which a scheduling policy is selected periodically from set of policies (a portfolio). For an intuition to this process, consider a synthetic workload that alternates two arrival patterns (Figure 3). For this favorable case, the portfolio scheduler adapts by selecting the appropriate scheduling policy. We have shown that portfolio schedulers generally outperform their constituent policies for compute-intensive many-task workloads. However, the data-intensive workloads and workloads with inter-job dependencies we consider in this work pose additional challenges, including different models of resource occupation and sharing, scheduling with dependencies, scheduling data and computation [22], etc., all in a variable-sized set of resources.

#### 4.2.3 Social awareness

The idea that social links between the users and between the providers of resources in a (data) processing system may influence the operation of the system dates from the early 1960s; for example, shared infrastructure generated off-line, socially guaranteed agreements between users, regarding equitable usage hours. However, recent studies have identified new forms of social influence on resource usage and sharing. The use of data by large groups of scientists has been shown to exhibit filecules, that is, sharing patterns for socially close participants [23]. Similarly, the use of computational resources in grids can be partially explained through social patterns [24]. A mid-2000s trend [25–27] of sharing resources based on socially guaranteed contracts led to many resources being gifted, volunteered, or shared. An important challenge is building mechanisms that exploit these social elements to improve resource usage and sharing. The challenge due to data sharing are non-trivial: data replication and transfers, incentives to manage foreign data, privacy-preserving processing of sensitive data [28]; early studies [29] have not yet addressed many of them.

### 4.3 Predictability challenges

For elastic systems, dynamic scheduling and dynamic resource provisioning are mandatory. However, in order to avoid overprovisioning and further resource wastage, understanding the performance of different workloads on different types of platforms is mandatory. The key research challenge here is to define a reliable methodology to predict a performance envelope of a (tasks,dataset,platform) tuple. We believe three important steps must be taken to advance the state-of-the-art in this direction: modeling, matchmaking, and benchmarking.

#### 4.3.1 Modeling

We find an important challenge in creating realistic performance envelopes for all major data processing stacks. We see this as an important challenge, which should extend the characterization of big data applications [16,17] towards more general models, where the performance envelope is a function of application (algorithm), dataset, data processing stack, and the (transient) resource substrate. To address this challenge, both workload modeling and platform modeling are necessary. Given the complexity of the analysis algorithms, as well as the large scale and heterogeneity of the platforms we use, none of these two challenges is getting any easier.

However, three important choices should make these challenges feasible by limiting the search space. First, we limit the modeling to data analytics applications, as we believe these algorithms will share a common set of characteristics that can be exploited to simplify modeling (e.g., they are all data-intensive, traversal-based, low-computation). Second, we include the dataset as part of the modeled workload, which should lower the abstraction level of the model. Third, and final, we recommend high-level symbolic platform models, calibrated using (micro-)benchmarking on real hardware.

#### 4.3.2 Matchmaking

When talking about models, matchmaking (or fitting) typically involves a combination of the workload and platform models that allows one reason about the expected performance. This prediction is very useful for dynamic scheduling, resource allocation, and especially resource prediction. Systematic and accurate matchmaking will avoid side effects such as overprovisioning and/or resource wastage.

We belive that, despite the complexity of the platform and workload models we expect to see, this matchmaking can be tackled if multiple levels of complexity and accuracy can be defined and accepted. As full modeling is likely to be too detailed (and time-consuming) for the purpose of dynamic scheduling and/or provisioning, it is the faster, higher level solutions that will be preferred for on-line scheduling, and the more accurate, detailed ones will be used for performance analysis and thorough prediciton.

#### 4.3.3 Benchmarking

In this context, benchmarking is required to calibrate and/or validate the workload and platform models. We recognize the significant lack of benchmarking strategies and suites in the filed of big data: although many early benchmarks have appeared for big data processing [30–33], many of them focused on the MapReduce programming model, and are limited with respect to the data flow diversity: data dependencies intra-query and intra-/inter-job, and large variations in the amount of the data transferred between tasks in the processing workflow.

For big data processing, we identify as a major challenge the formulation of realistic, cost- and time-effective, fair benchmarking suites [34]. Similarly to the approach taken by TPC, such benchmarks should focus on broad application domains, such as graph-processing [4, 35] or time-based analytics [36].

We note that the benchmarking results can be used for designing or calibrating realistic, yet tunable models of workloads [37], with important contributions in modeling and generation of large datasets [38, 39] and abstract workload modeling (e.g., statistical analysis on multi-workload data archives [40]) .

### 4.4 Compatibility / Interfacing Challenges

By decoupling the execution engine, we have been able to dive into its components and analyze the improvements that

can be made for performance gain. However, the same "extraction" of this engine from the monolithic stucture of the data processing stack has posed an additonal challenge: clear specifications for interfacing with the rest of the architecture are now needed. We focus here on two of these specifications: the "north" interface, which relates to the task-based application specification, and the "south" interface, which relates to the storage system.

### 4.4.1 Interfacing with the application

Our model assumes that the application is eventually specified as a large collection of tasks. Ideally, they should be independent. The more dependent (i.e., tightly coupled) they are, the most difficult it becomes to specify their interaction.

Essentially, this interface requires matching the back-end of the high level programming model with the front-end of the execution engine. To insure this match is done by construction (i.e., to avoid yet another intermediate layer for transposition), more fundamental research is requried into task specification, characterization, and classification. Ultimately, this will also affect the design of high level programming models, but this remains beyond the scope of this work.

### 4.4.2 Storage management

For an application to be described as a collection of tasks, the entire workload (i.e., algorithm and data) needs to be decomposed. While the "migration" of the computation depending on the available resources is the responsibility of the execution engine, the similar migration of the data requires interacting with the storage system.

We believe there is much research to be put into optimizing this interaction for two reasons: (1) in elastic and dynamic systems, such migrations happen very often, and (2) the expected heterogeneity of the platforms will force migration between different storage spaces, depending on the source and destination. They key challenges to address here are understanding the types of data migration and characterizing the performance penalty each type infers. Based on this classification, data migration rules can be enforced on the interface between the execution and storage engines, thus limiting the negative performance impact due to too often/too expensive migrations.

## 5. CONCLUSION

Big data processing has been increasing in popularity for the past five years. Smart and efficient data analytics become important revenue sources for many small and medium enterprises (SMEs), which have the expertise for interesting analysis, but no resources to fund expensive infrastructure. We belive that despite relying on commodity hardware and/or cloud services, high performance remains achievable for these common scenarios, but it is currently hindered by the monolithic nature of existing big data processing solutions.

Thus, we proposed our vision of an open architecture for big data processing, focusing on its execution engine, the component with the highest performance impact. Our contributions are threefold: (1) describing a vision for an open, non-monolithic big data processing architecture, (2) identifying its main design requirements, and (3) enumerating the most important eleven challenges that need to be addressed to transform this vision into a feasible solution.

Our own work addresses some of these challenges in more detail. For example, for the performance challenges, we are actively involved in designing and evaluating portable parallel programming models for multi- and many-core architectures [8, 41], as well as in addressing the impact of heterogeneity on performance [9, 42]. We are also working on the problem of predictability - especially on its modeling and benchmarking components - in our work on large scale graph processing on multiple platforms [4, 42]. Finally, our work on scheduling in large scale distributed environments and clouds is addressing the elasticity challenges [19, 20].

All our research efforts in this directions are, of course, only the tip of the iceberg, proving that solutions do exist for improving the state-of-the-art, but much more needs to be done to fully materialize our vision of an open big data processing architecture, with a flexible, performance-aware execution engine that will allow regular users (SMEs) to efficiently use the resources available to them (off-the-shelf or from a cloud, homogeneous or heterogeneous) to run their analytics.

## Acknowledgment

The authors are grateful for the comments of the reviewers. This publication is supported by the Dutch national program COMMIT, and the STW/NWO Veni grants LarGe (11881) and Graphitti (12480).

## 6. REFERENCES

[1] V. R. Borkar and M. J. Carey, "Big data technologies circa 2012," in *COMAD*, 2012, pp. 12–14.

[2] R. Ramakrishnan, "Big data in 10 years," in *IPDPS*, 2013, p. 887.

[3] V. R. Borkar, M. J. Carey, and C. Li, "Big data platforms: what's next?" *ACM Crossroads*, vol. 19, no. 1, pp. 44–49, 2012.

[4] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke, "How well do graph-processing platforms perform? an empirical performance evaluation and analysis: Extended report," Delft University of Technology, Tech. Rep. PDS-2013-004, 2013. [Online]. Available: http://www.pds.ewi.tudelft.nl/fileadmin/pds/reports/2013/PDS-2013-004.pdf

[5] M. Stonebraker and J. Robertson, "Big data is 'buzzword du jour;' cs academics 'have the best job'," *Commun. ACM*, vol. 56, no. 9, pp. 10–11, 2013.

[6] M. Stonebraker and J. Hong, "Researchers' big data crisis; understanding design and functionality," *Commun. ACM*, vol. 55, no. 2, pp. 10–11, 2012.

[7] R. Ramakrishnan, "Cap and cloud data management," *IEEE Computer*, vol. 45, no. 2, pp. 43–49, 2012.

[8] J. Shen, J. Fang, H. Sips, and A. L. Varbanescu, "Performance Traps in OpenCL for CPUs," in *PDP 2013*, February 2013, pp. 38–45.

[9] J. Shen, A. L. Varbanescu, H. J. Sips, M. Arntzen, and D. G. Simons, "Glinda: a framework for accelerating imbalanced applications on heterogeneous

platforms," in *Conf. Computing Frontiers*, 2013, p. 14.
[10] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *SIGMOD*, 2008, pp. 1099–.
[11] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a High-Level Dataflow System on top of Map-Reduce: the Pig Experience," *Proc. of the VLDB Endowment*, vol. 2, no. 2, pp. 1414–1425, 2009.
[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Fast and interactive analytics over Hadoop data with Spark," *;login*, vol. 37, no. 4, 2012.
[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *MSST*. IEEE, 2010, pp. 1–10.
[14] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds," in *CCGRID*, 2012, pp. 612–619.
[15] B. Ghit, N. Yigitbasi, and D. H. J. Epema, "Resource management for dynamic mapreduce clusters in multicluster systems," in *SC Companion*, 2012, pp. 1252–1259, mTAGS, Best Paper Award.
[16] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009.
[17] A. S. van Amesfoort, A. L. Varbanescu, and H. J. Sips, "Parallel application characterization with quantitative metrics," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 5, pp. 445–462, 2012.
[18] A. Gharaibeh, L. B. Costa, E. Santos-Neto, and M. Ripeanu, "A yoke of oxen and a thousand chickens for heavy lifting graph processing," in *PACT*, 2012.
[19] S. Shen, K. Deng, A. Iosup, and D. H. J. Epema, "Scheduling jobs in the cloud using on-demand and reserved instances," in *Euro-Par*, 2013, pp. 242–254.
[20] K. Deng, R. Verboon, K. Ren, and A. Iosup, "A periodic portfolio scheduler for scientific computing in the data center," in *JSSPP*, 2013, To Appear.
[21] K. Deng, J. Song, K. Ren, and A. Iosup, "Exploring portfolio scheduling for long-term execution of scientific workloads in IaaS clouds," in *SC*, 2013.
[22] K. Ranganathan and I. T. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *HPDC*, 2002, pp. 352–358.
[23] S. Doraimani and A. Iamnitchi, "File grouping for scientific data management: lessons from experimenting with real traces," in *HPDC*, 2008, pp. 153–164.
[24] S. Zheng, Z.-Y. Shae, X. Zhang, H. Jamjoom, and L. Fong, "Analysis and modeling of social influence in high performance computing workloads," in *Euro-Par (1)*, 2011, pp. 193–204.
[25] K. Ranganathan, M. Ripeanu, A. Sarin, and I. T. Foster, "Incentive mechanisms for large collaborative resource sharing," in *CCGRID*, 2004, pp. 1–8.
[26] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *GRID*, 2004, pp. 4–10.
[27] W. Cirne, F. V. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray, "Labs of the world, unite!!!" *J. Grid Comput.*, vol. 4, no. 3, pp. 225–246, 2006.
[28] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Pérez-González, "Privacy-preserving data aggregation in smart metering systems: An overview," *IEEE Signal Process. Mag.*, vol. 30, no. 2, pp. 75–86, 2013.
[29] H. Lin, X. Ma, J. S. Archuleta, W. chun Feng, M. K. Gardner, and Z. Zhang, "Moon: Mapreduce on opportunistic environments," in *HPDC*, 2010, pp. 95–106.
[30] K. Kim, K. Jeon, H. Han, S.-g. Kim, H. Jung, and H. Y. Yeom, "MRBench: A Benchmark for MapReduce Framework," in *ICPADS*. IEEE, 2008, pp. 11–18.
[31] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The Hibench Benchmark Suite: Characterization of the MapReduce-based Data Analysis," *ICDE Workshops*, pp. 41–51, 2010.
[32] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites," in *MASCOTS*. IEEE, 2011, pp. 390–399.
[33] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "PUMA: Purdue MapReduce Benchmarks Suite," 2012.
[34] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun, "Benchmarking in the cloud: What it should, can, and cannot be," in *TPCTC*, 2012, pp. 173–188.
[35] B. Elser and A. Montresor, "An evaluation study of bigdata frameworks for graph processing," in *IEEE Big Data*, 2013, To Appear.
[36] T. Hegeman, B. Ghit, M. Capota, J. Hidders, D. Epema, and A. Iosup, "The BTWorld use case for big data analytics: Description, mapreduce logical workflow, and empirical evaluation," in *IEEE Big Data*, 2013, To Appear, [Online] Available: http://www.pds.ewi.tudelft.nl/~iosup/btworld-mapreduce-workflow13ieeebigdata.pdf.
[37] M. Stonebraker, "A new direction for tpc?" in *TPCTC*, 2009, pp. 11–17.
[38] A. Alexandrov, K. Tzoumas, and V. Markl, "Myriad: Scalable and expressive data generation," *PVLDB*, vol. 5, no. 12, pp. 1890–1893, 2012.
[39] M.-D. Pham, P. A. Boncz, and O. Erling, "S3g2: A scalable structure-correlated social graph generator," in *TPCTC*, 2012, pp. 156–172.
[40] A. Iosup and D. H. J. Epema, "Grid computing workloads," *IEEE Internet Computing*, vol. 15, no. 2, pp. 19–26, 2011.
[41] A. L. Varbanescu, P. Hijma, R. van Nieuwpoort, and H. Bal, "Towards an effective unified programming model for many-cores," *APDCM*, 2011.
[42] A. Penders, "Accelerating graph analysis with heterogeneous systems," Master's thesis, TUDelft, December 2012.