A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures

Shantenu Jha[1], Judy Qiu[2], Andre Luckow[1], Pradeep Mantha[1], Geoffrey C.Fox[2*]
(1) *RADICAL, Rutgers University, Piscataway, NJ 08854, USA* (2) Indiana University, USA
(*) *Contact Author:gcf@indiana.edu*

***Abstract*—Scientific problems that depend on processing large amounts of data require overcoming challenges in multiple areas: managing large-scale data distribution, co-placement and scheduling of data with compute resources, and storing and transferring large volumes of data. We analyze the ecosystems of the two prominent paradigms for data-intensive applications, hereafter referred to as the high-performance computing and the Apache-Hadoop paradigm. We propose a basis, common terminology and functional factors upon which to analyze the two approaches of both paradigms. We discuss the concept of "Big Data Ogres" and their facets as means of understanding and characterizing the most common application workloads found across the two paradigms. We then discuss the salient features of the two paradigms, and compare and contrast the two approaches. Specifically, we examine common implementation/approaches of these paradigms, shed light upon the reasons for their current "architecture" and discuss some typical workloads that utilize them. In spite of the significant software distinctions, we believe there is architectural similarity. We discuss the potential integration of different implementations, across the different levels and components. Our comparison progresses from a fully qualitative examination of the two paradigms, to a semi-quantitative methodology. We use a simple and broadly used Ogre (K-means clustering), characterize its performance on a range of representative platforms, covering several implementations from both paradigms. Our experiments provide an insight into the relative strengths of the two paradigms. We propose that the set of Ogres will serve as a benchmark to evaluate the two paradigms along different dimensions.**

I. Introduction

The growing importance of data-intensive applications is generally recognized and has lead to a wide range of approaches and solutions for data distribution, management and processing. These approaches are characterized by a broad set of tools, software frameworks and implementations. Although seemingly unrelated, the approaches can be better understood by examining their use of common abstractions and similar architectures for data management and processing. Building upon this putative similarly, we examine and organize many existing approaches to Big Data processing into two primary paradigms – the scientific high-performance (HPC) and Apache-Hadoop paradigms, which we believe reflects and captures the dominant historical, technical and social forces that have shaped the landscape of Big Data analytics.

The HPC paradigm has its roots in supercomputing-class computationally intensive scientific problems (e.g. Molecular Dynamics of macromolecular systems, fluid dynamics at scales to capture turbulence) and in managing large-scale distributed problems (e. g. data analysis from the LHC). HPC paradigm has been characterized by limited implementations, but customized and tuned for performance along a narrow set of requirements. In contrast, the Apache-Hadoop paradigm, here after referred to simply as the Apache Big Data Stack (ABDS) has seen a significant update in industry and recently also in scientific environments. A vibrant, manifold open-source ecosystem consisting of higher-level data stores, data processing/analytics and machine learning frameworks has evolved around a stable, non-monolithic kernel: the Hadoop Filesystem (HDFS) and YARN. Hadoop integrates compute and data, and introduces application-level scheduling as a means to facilitate heterogeneous application workloads and high cluster utilization.

The success and evolution of ABDS into a widely deployed cluster computing frameworks yields many opportunities for *traditional* scientific applications; it also raises many important questions, viz., What features of Hadoop are useful for traditional scientific workloads? What features of the ABDS can be extended and integrated with the HPC implementations? How do typical data-intensive HPC and ABDS workloads differ? It is currently difficult for most applications to utilize the two paradigms interoperably. The divergence and heterogeneity will likely increase due to the continuing evolution of ABDS, thus we believe it is important and timely to answer questions that will support interoperable approaches. However, before such interoperable approaches can be formulated, it is important to understand the different abstractions, architectures and applications that each paradigm utilizes and supports. It is the aim of this paper to provide the conceptual framework and terminology, so as to begin addressing questions of interoperability.

Paper Outline: This paper is divided into two logical parts: in the first, we analyze the ecosystem of the two primary paradigms to data-intensive applications. We discuss the salient features of the two paradigms, compare and contrast the two for functionality and implementations along the layers of analysis, runtime-environments, communication layer, resource management layer and the physical resource layer. In the second part, we move from a fully qualitative examination of the two, to a semi-quantitative methodology, whereby we experimentally examine both hard performance numbers (along different implementations of the two stacks) and soft issues such as completeness, expressivity, extensibility as well as software engineering considerations.

II. Data-Intensive Application: Big Data Ogres

Based upon an analysis of a large set of Big Data applications, including more than 50 use cases [1], we propose the Big Data Ogres in analogy with parallel computing with the Berkeley Dwarfs, NAS benchmarks and linear algebra templates. The purpose of Big Data Ogres is to discern commonalities and patterns across a broad range of seemingly different Big Data applications, propose an initial structure to classify them, and help cluster some commonly found applications using structure. Similar to the Berkeley Dwarfs, the Big Data Ogres are not orthogonal, nor exclusive, and thus do not constitute a formal taxonomy. We propose the Ogres as a benchmark to investigate and evaluate the paradigms for architectural principles, capabilities and implementation performance. Also we capture the richness of Big Data by including not just different parallel structures but also important overall patterns. Big Data is in its infancy without clear consensus as to important issues and so we propose an inclusive set of Ogres expecting that further discussion will refine them.

The first facet captures **different problem architectures**. Some representative examples are (i) Pleasingly Parallel – as in Blast (over sequences), Protein docking (over proteins and docking sites), imagery, (ii) Local Machine Learning (ML) – or filtering pleasingly parallel as in bio-imagery, radar (this contrasts with Global Machine Learning seen in LDA, Clustering etc. with parallel ML over nodes of system), (iii) Fusion – where knowledge discovery often involves fusion of multiple methods (ensemble methods are one approach), (iv) Data points in metric or non-metric spaces, (v) Maximum Likelihood, (vi) χ^2minimizations, (vii) Expectation Maximization (often Steepest descent), and (viii) Quantitative measures for Big Data applications which can be captured by absolute sizes and relative ratios of flops, IO bytes and communication bytes.

The second facet captures applications with **important data sources with distinctive features**, representative examples of the data sources include, (i) SQL based, (ii) NOSQL based, (iii) Other enterprise data systems, (iv) Set of Files (as managed in iRODS), (v) Internet of Things, (vi) Streaming, (vii) HPC simulations, and (viii) Temporal features – for in addition to the system issues, there is a temporal element before data gets to compute system, e.g., there is often an initial data gathering phase which is characterized by a block size and timing. Block size varies from month (Remote Sensing, Seismic) to day (genomic) to seconds (Real time control, streaming)

The third facet contains **Ogres themselves classifying core analytics and kernels/mini-applications/skeletons**, with representative examples (i) Recommender Systems (Collaborative Filtering) (ii) SVM and Linear Classifiers (Bayes, Random Forests), (iii) Outlier Detection (iORCA) (iv) Clustering (many methods), (v) PageRank, (vi) LDA (Latent Dirichlet Allocation), (vii) PLSI (Probabilistic Latent Semantic Indexing), (viii) SVD (Singular Value Decomposition), (ix) MDS (Multidimensional Scaling), (x) Graph Algorithms (seen in neural nets, search of RDF Triple stores), (xi) Neural Networks (Deep Learning), (xii) Global Optimization (Variational Bayes), (xiii) Agents, as in epidemiology (swarm approaches) and (xiv) GIS (Geographical Information Systems).

III. Architecture and Abstractions: HPC and ABDS Ecosystems

In this section we compare and contrast the ABDS and HPC ecosystems, viz. the underlying architectural assumptions, the primary abstractions (both conceptual and implementation). Figure 1 depicts the different layers and architectural design approaches and highlights some of the primary abstractions. For the purpose of our comparison we identified five layers: resource fabric, resource management, communication, higher-level runtime environment and data processing/analytics. HPC infrastructure were traditionally built for scientific applications aimed toward high-end computing capabilities (small input, large output). Hadoop in contrast was built to process large volumes of data (large input, small output), resulting in different software stacks.

A. High Performance Computing

In a typical HPC cluster compute and data infrastructures are separated: A high-end compute environment – typically a shared nothing many-core environment (potentially adding GPUs or other accelerators such as the Xeon Phi) – is complemented by a storage cluster running Lustre GPFS [2] or another parallel filesystem connected by a high-bandwidth, low-latency network. While this meets the need for compute-intensive applications, for data-intensive applications this means that data needs to be moved across the network, which represents a potential bottleneck. Compute resources are typically managed by a local resource management system such as SLURM, Torque or SGE. Generally, these system have a focus on managing compute slots (typically cores).

Compute resources are a typically managed by a local resource management system such as SLURM, Torque or SGE. Generally, these system have a focus on managing compute slots (typically cores). Storage resources in HPC are shared resources, where a quota is applied on the data size, but not on I/O. Data locality and other scheduling constraints are typically not considered. Lustre and GPFS storage resources are typically exposed as shared filesystem on the compute nodes. In addition several specialized higher-level storage management services, such as SRM [3], iRODS [4] have emerged. iRODS is a comprehensive distributed data management solution designed to operate across geographically distributed, federated storage resources. iRODS combines storage services with services for metadata, replica, transfer management and scheduling. In contrast to ABDS, data-management in HPC is typically done using files and not using higher level abstractions as found in the Hadoop ecosystem.

Various other approaches for supporting data-intensive applications on the HPC infrastructures emerged. For example, different MapReduce implementations for HPC have been proposed: MPI-based MapReduce implementations, such as MapReduce-MPI [5], can efficiently utilize HPC features, such as low-latency interconnects and one-sided

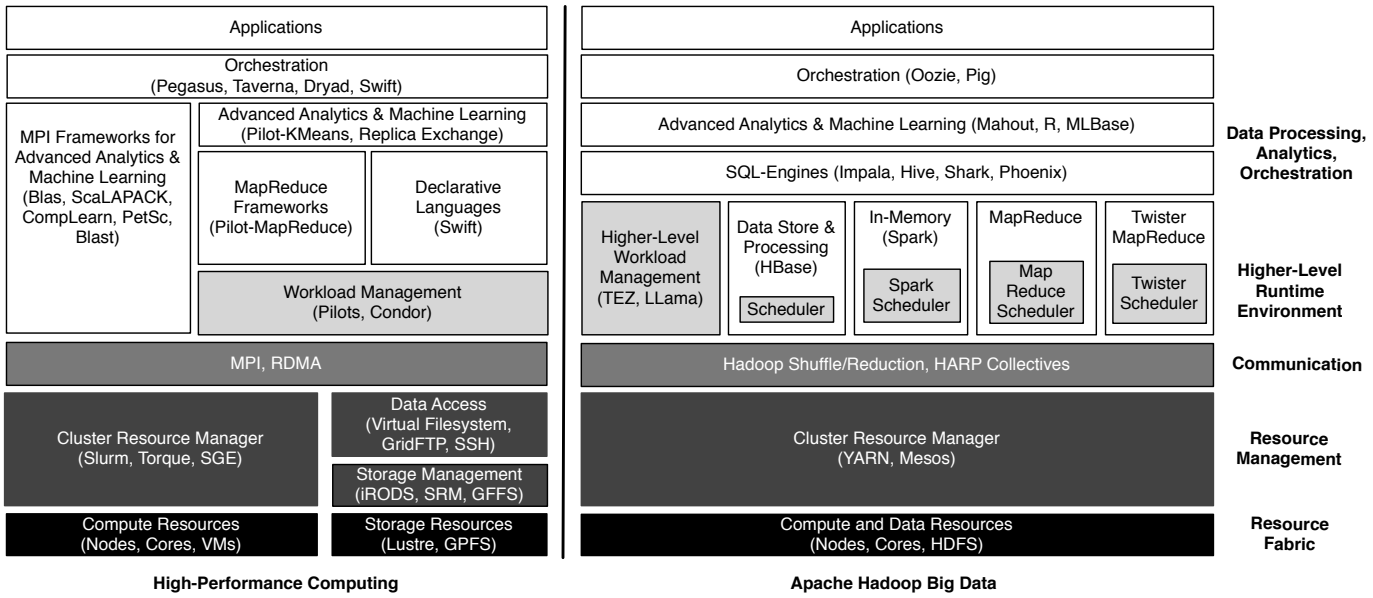

Fig. 1. **HPC and ABDS architecture and abstractions:** The HPC approach historically separated data and compute; ABDS co-locates compute and data. The YARN resource manager heavily utilizes multi-level, data-aware scheduling and supports a vibrant Hadoop-based ecosystem of data processing, analytics and machine learning frameworks. Each approach has a rich, but hitherto distinct resource management and communication capabilities.

and non-blocking communications [6]. Further, various non-MPI MapReduce implementations have been proposed: Twister/Salsa [7] to support iterative machine learning workloads, Pilot-MapReduce [8] to support geographically distributed data, etc.

In addition several runtime environments for supporting heterogeneous, loosely coupled tasks, e. g. Pilot-Jobs [9], many tasks [10] and workflows [11]. Pilot-Jobs generalize the concept of a placeholder to provide multi-level and/or application-level scheduling on top of the system-provided schedulers. With the increasing importance of data, Pilot-Jobs are increasingly used to process and analyze large amounts of data [12], [9]. In general, one can distinguish two kinds of data management: (i) the ability to stage-in/stage-out files from another compute node or a storage backend, such as SRM and (ii) the provisioning of integrated data/compute management mechanisms. An example for (i) is Condor-G/Glide-in [13], which provides a basic mechanism for file staging and also supports access to SRM storage. DIRAC [14] is an example of a type (ii) system providing more integrated capabilities.

B. ABDS Ecosystem

Hadoop was originally developed in the enterprise space (by Yahoo!) and introducing an integrated compute and data infrastructure. Hadoop provides an open source implementation of the MapReduce programming model originally proposed by Google [15]. Hadoop is designed for cheap commodity hardware (which potentially can fail), co-places compute and data on the same node and is highly optimized for sequential reads workloads. With the uptake of Hadoop in the commercial space, scientific applications and infrastructure providers started to evaluate Hadoop for their purposes. At the same time, Hadoop evolved with increasing requirements (e. g. the support for very heterogeneous workloads) into a general purpose cluster framework borrowing concepts existing in HPC.

Hadoop-1 had two primary components (i) the Hadoop Filesystem [16] – an open source implementation of the Google Filesystem architecture [17] – and (ii) the MapReduce framework which was the primary way of parallel processing data stored in HDFS. However, Hadoop saw a broad uptake and the MapReduce model as sole processing model proofed insufficient. The tight coupling between HDFS, resource management and the MapReduce programming model was deemed to be too inflexible for the usage modes that emerged in the Hadoop ecosystem. An example of such a deficit is the lack of support for efficient iterative computations (as often found in machine learning). With the introduction of Hadoop-2 and YARN [18] as central resource manager, Hadoop clusters can now accommodate any application or framework. As shown in Figure 1 (right) a vibrant ecosystem of higher-level runtime systems, data processing and machine learning libraries emerged on top of resource fabric and management layers, i. e. HDFS and YARN. Historically, MapReduce was the Hadoop runtime layer for processing data; but, in response to application requirements, runtimes for record-based, random-access data (HBase [19]), iterative processing (Spark [20], TEZ [21], Twister [7]), stream (Spark Streaming) and graph processing (Apache Giraph [22]) emerged. A key enabler for these frameworks is the YARN support for multi-level scheduling, which enables the application to deploy their own application-level scheduling routines on top of Hadoop-managed storage and compute resources. While YARN manages the lower resources, the higher-level runtimes typically use an application-

level scheduler to optimize resource usage for the application. In contrast to HPC, the resource manager, runtime system and application are much more tighter integrated. Typically, an application uses the abstraction provided by the runtime system (e. g. MapReduce) and does not directly interact with resource management.

Spark is a runtime for iterative processing; it is based on a Scala-based API for expressing parallel dataflow on top of in-memory, distributed datasets. For this purpose, Spark introduces resilient distributed datasets (RDD) as higher-level API that enables application to load a dataset into the memory of a set of cluster nodes. The runtime of Spark automatically partitions the data and manages data locality during runtime.

Many analytical applications – in particular in enterprise environments – rely on SQL as data query language. This led to the development of several SQL-based analytic engines in the data processing layer: Google's Dremel [23], Hive [24], HAWQ [25], Impala [26] and Shark [27] are examples of such engines. While Hive was originally implemented based on the MapReduce model, the latest version relies on TEZ as runtime layer. Similarly, Shark relies on Spark as runtime. Other SQL engines, such as Impala and HAWQ, provide their own runtime environment and do not rely on MapReduce or Spark. In addition, hybrid relational database/HDFS environments have been proposed. HadoopDB [28] e. g. deploys a PostgreSQL database on every node on which it distributes the data using hash partition. Further, it can access data from HDFS via external tables. Several commercial approaches with similar features exist, e. g. Polybase [29].

Advanced Analytics & Machine Learning layer applications/frameworks typically require multiple iterations on the data. While traditional in-memory, single-node tools, such R [30] or Scikit-Learn[31] provide powerful implementations of many machine learning algorithms, they are mostly constrained to a single machine. To overcome this limitation, several approaches that rely on scalable runtime environments in the Hadoop ecosystem have been proposed, e. g. Apache Mahout [32] or RHadoop [33]. There are well-known limitations of Hadoop-1 with respect to support for iterative MapReduce applications [7]. Thus, increasingly, iterative runtimes are used for advanced analytics. MLBase [34] is a machine learning framework based on Spark as lower-level data processing framework. SparkR [35] allows R applications to utilize Spark.

C. Resource Management

Table I summarizes different architectures for resource management: centralized, multi-level and decentralized. HPC schedulers are centralized and designed for rigid applications with constant resource requirements. HPC applications, such as large, monolithic simulations spawning a large number of cores, typically utilize tightly-coupled, often MPI-based parallelism. MPI applications are tightly-coupled and highly latency sensitive. While these applications have fixed resource demands (with respect to number of cores, memory and walltime), which does not change during the lifetime, they need to be scheduled in a way that they simultaneously execute on a system e. g. using Gang scheduling. Scheduling is done on job-level, i. e. application-level tasks (e. g. the execution of the individual processes on the compute nodes) are not exposed to the resource manager. Thus, scheduling heterogeneous workload consisting of small, short-running tasks and longer running batch-oriented tasks represents a challenge for traditional monolithic, centralized cluster scheduling systems. Often, Pilot-Jobs are used to overcome the flexibility limitations and support dynamic applications comprising of heterogeneous tasks. Data locality is not a primary concern in HPC: most HPC applications are write heavy, while data-intensive applications are read heavy.

	Centralized	**Multi-Level**	**Decentralized**
Examples	Torque, SLURM	YARN, Mesos, Pilots	Omega, Sparrow
Workloads	large, parallel jobs	medium-sized tasks	fine-grained tasks
Latency	high	medium - low	low
Application Integration	Submission only	Application-Level Scheduling	Application-Level Scheduling

TABLE I
SCHEDULER ARCHITECTURES: CENTRALIZED, MONOLITHIC RESOURCE MANAGERS ARE BEING REPLACED WITH MORE SCALABLE ARCHITECTURES EMPHASIZING APPLICATION-LEVEL SCHEDULING.

Data-intensive workloads in contrast can be decomposed into fine-grained, loosely-coupled parallel tasks. By scheduling on task-level rather than on job-level, the utilization of resources and fairness can be improved [36]. Multi-level schedulers, such as YARN [18] and Mesos [37], efficiently support data-intensive workloads comprised a data-parallel, loosely-coupled tasks and allow the dynamic allocation and usage of resource through application-level scheduling. Decentral schedulers aim to address scalability bottlenecks and low latency requirements of interactive workloads. Google's Omega [38] and Sparrow [36], an application-level scheduler for Spark, are examples of decentral schedulers. In the following we focus on investigate the evolution of the Hadoop schedulers focusing on the central and multi-level approaches.

Hadoop-1 utilizes a centralized scheduling approach using the Job Tracker as resource manager. Not only represented the Job Tracker a scalability bottleneck, it also tightly coupled the MapReduce framework significantly constraining flexibility. In particular in the early days this was not an issue: Hadoop was often used on top of HPC clusters using Hadoop on Demand [39], SAGA Hadoop [40] or MyHadoop [41] or in clouds (Amazon's Elastic MapReduce [42] or Microsoft's HDInsight [43]). A limitation of these approaches is the lack of data locality and thus, the necessity to initially move data to the HDFS filesystem before running the computation.

Despite the limitations of Hadoop-1, many different resources usage modes for Hadoop clusters emerged. However, with the increasing size and variety of frameworks and applications, the requirements with respect to resource management increased, e. g. it became a necessity to support batch, streaming and interactive data processing. Often, higher-level frameworks, such as HBase or Spark, were deployed next to the core Hadoop daemons making it increasingly difficult to predict resource usage and thus, performance. YARN [18], the core of Hadoop-2, was designed to address this need and to

efficiently support heterogeneous workloads in larger Hadoop environments. Another multi-level scheduler for Hadoop is Mesos [37]; While there are some mostly syntactic differences – Mesos e. g. uses a resource offer model, while YARN uses resource requests – it is very similar to YARN providing multi-level scheduling for heterogeneous workloads.

An increasingly larger ecosystem evolved on top HDFS and YARN. As shown in Figure 1 applications frameworks typically rely on a runtime system that embeds an application-level scheduler that tightly integrates with YARN (e. g. MapReduce, Spark and HBase all provide their application-level scheduler). Increasingly, common requirements are integrated into higher-level, shared runtime systems frameworks, e. g. the support for long-running or interactive applications or multi-stage applications using DAGs (directed acyclic graph). For example, Llama [44] offers a long-running application master for YARN application designed for the Impala SQL engine. TEZ [21] is a DAG processing engine primarily designed to support the Hive SQL engine.

As described, typical data-intensive workloads consist of short-running, data-parallel tasks; By scheduling on task-level instead on job-level, schedulers, such as YARN, are able to improve the overall cluster utilization by dynamically shifting resources between application. The scheduler can e. g. easily remove resources from an application by simply waiting until task completion. To enable this form of dynamic resource usage, YARN requires a tighter integration of the application. The application e. g. needs to register an Application Master process, which subscribes to a set of defined callbacks. The unit of scheduling is referred to as a container. Containers are requested from the Resource Manager. In contrast to HPC schedulers, the Resource Manager does not necessarily return the requested number of resources, i. e. the application is required to elastically utilize resources as they become allocated by YARN; Also, YARN can request the de-allocation of containers requiring the application to keep track of currently available resources.

D. High-Performance Big Data Stack: A Convergence of Paradigms?

While HPC and Hadoop were originally designed to support different kinds of workloads: high-end, parallel computing in the HPC case versus cheap data storage and retrieval in the Hadoop case, a convergence at many levels can be observed. Increasingly, more compute-demanding workloads are deployed on Hadoop cluster, while more data-parallel tasks and workflows are executed on HPC infrastructures. With the introduction of YARN and Mesos, the Hadoop ecosystem has matured to support a wide range of heterogeneous workloads. At the same time a proliferation of tools (e. g. Pilot-Jobs) to support loosely-coupled, data-intensive workloads on HPC infrastructures emerged. However, these tools often focus on supporting large number of compute tasks or are constraint to specific domains; thus, they do not reach the scalability and diversity of the Hadoop ecosystem.

The ABDS ecosystem provides a wide-range of higher-level abstractions for data storage, processing and analytics (MapReduce, iterative MapReduce, graph analytics, machine learning etc.) all built on top of extensible kernels: HDFS and YARN. In contrast to HPC schedulers, a first-order design objective of YARN is the support for heterogeneous workloads using multi-level, data-aware scheduling. For this purpose, YARN requires a higher degree of integration between the application/framework and the system-level scheduler than typical HPC schedulers. Instead of a static resource request prior to the run, YARN applications continuously request and return resources in a very fine-grained way, i. e. applications can optimize their resource usage and the overall cluster utilization is improved.

While YARN is currently not an option for HPC resource fabrics, trends and demonstrated advantages have lead to proposals for integrating Hadoop/YARN and HPC. The following aspects need to be addressed: (i) integration with the local resource management level system, (ii) integration with HPC storage resources (i. e. the shared, parallel filesystem) and (iii) the usage of high-end network features such as RDMA and efficient abstractions (e. g. collective operations) on top of these.

Resource Management Integration: To achieve integration with the native, system-level resource management system, the Hadoop-level scheduler can be deployed on top of the system-level scheduler. Resource managers, such as Condor and SLURM, provide Hadoop support. Further, various third-party systems, such as SAGA-Hadoop [40], JUMMP [45] or MyHadoop [41], exist. A main disadvantage with this approach is the loss of data-locality, which the system-level scheduler is typically not aware of. Also, if HDFS is used, data first needs to be copied into HDFS before it can be processed, which represents a significant overhead. Further, these systems use Hadoop in a single user mode; thus, cluster resources are not used in an optimal way.

Storage Integration: Hadoop provides a pluggable filesystem abstraction that interoperates with any Posix compliant filesystem. Thus, most parallel filesystems can easily be used in conjunction with Hadoop; however, in these cases the Hadoop layer will not be aware of the data locality maintained on the parallel filesystem level, e. g. Intel supports Hadoop on top of Lustre [46], IBM on GPFS [47]. Another optimization concerns the MapReduce shuffling phase that is carried out via the shared filesystem [48]. Also, the scalability of these filesystem is usually constraint compared to HDFS, where much of the data processing is done local to the compute avoiding data movements across the network. Thus, HDFS is less reliant on fast interconnects.

Network Integration: Hadoop was designed for Ethernet environments and mainly utilizes Java sockets for communications. Thus, high-performance features such as RDMA are not utilized. To address this issue, RDMA support in conjunction with several optimizations to HDFS, MapReduce, HBase and other components for Infiniband or 10 Gigabit networks has been proposed [49]. HARP introduces an abstraction for collective operations with Hadoop jobs [50].

Implementation	Execution Unit	Data Model	Intermediate Data Handling	Resource Management	Language	Hardware
(A.1) Hadoop	Process	Key, Value pairs (Java Object)	Disc/Local (and network)	YARN	Java	HPC Madrid: 16 cores/node, 16 GB memory, GE
(A.2) Mahout	Process	Mahout Vectors	Disc (and network)	Hadoop Job Tracker	Java	EC2: cc1.4xlarge, 16 cores, 23 GB memory, 10GE
(B) MPI	Process (long running)	Primitive Types, Arrays	Message Passing (network)	Amazon/ mpiexec	C	EC2: cc1.4xlarge, 16 cores/node, 23 GB memory, 10GE
(C.1) Python-Script (Pilot-KMeans)	Process	Key/Value (Text)	Disk/Lustre	Pilots, SLURM	Python, Java	HPC Stampede: 16 cores/node, 32 GB memory, Infiniband
(C.2) HARP	Thread (long running)	Key/Value (Java Object)	Collectives (network)	YARN	Java	HPC Madrid: 16 cores/node, 16 GB memory, GE
(C.3) Spark	Thread	Key/Value (RDD)	Spark Collectives (network)	YARN	Java, Scala	EC2: cc1.4xlarge, 16 cores/node, 23 GB memory, 10GE

TABLE II
K-MEANS – COMPARISON OF DIFFERENT IMPLEMENTATIONS AND INFRASTRUCTURES

IV. EXPERIMENTS

In the following we run different implementations of one of the Ogres of Section II; we choose the K-Means clustering algorithm. Table II summarizes these different implementations. We categorize these into three categories: (A) Hadoop, (B) HPC and (C) hybrid implementations. For (A) we investigate (A.1) an Hadoop MapReduce implementation and (A.2) Apache Mahout [32]; for (B) an MPI-based K-Means implementation [51]. We examine the following hybrid approaches: (C.1) Python Scripting implementation using Pilots [8] (Pilot-KMeans), (C.2) a Spark K-Means [52] and (C.3) a HARP implementation [50]. While (C.1) provides an interoperable implementation of the MapReduce programming model for HPC environments, (C.2) and (C.3) enhance Hadoop for efficient iterative computations and introduce collective operations to Hadoop environments.

We use Amazon EC2, the Madrid YARN/Hadoop cluster and the Stampede clusters (which is part of XSEDE [53]) as the different resources. On EC2 we utilize the cluster compute instance type, which provides a HPC-style environment. We utilize Elastic MapReduce [54] for managing the Hadoop cluster in scenario (A.1) and the `spark-ec2` tool for scenario (C.3). Madrid uses YARN as resource manager; SLURM is deployed on Stampede. We run three different K-Means scenarios: (i) 1,000,000 points and 50,000 clusters, (ii) 10,000,000 points and 5,000 clusters and (iii) 100,000,000 points and 500 clusters. Each K-Means iteration comprises of two phases that naturally map to the MapReduce programming model: in the map phase the closest centroid for each point is computed; in the reduce phase the new centroids are computed as the average of all points assigned to this centroid. While the computational complexity is defined by the number of points $\times$ number of clusters (and thereby a constant in the aforementioned scenarios), the amount of data that needs to be exchanged during the shuffle phase increases gradually from scenario (i) to (iii), in proportion to the number of points.

Figure 2 shows the results of experiments. Both Hadoop implementations of K-Means (Hadoop MR (A.1)/Mahout (A.2)) perform significantly worse than the MPI-based implementation. The Map Reduce model – the predominant usage mode of Hadoop-1 – has several disadvantages with respect to supporting iterative machine learning algorithms: The shuffle phase, i. e. the sorting of the output keys and the movement of the data to the reduce task, is optimized for use cases, such as data grouping, but introduces a significant overhead where sorting is not needed. In the case of larger amounts of shuffle data, data needs to be spilled to disks; sorting is not essential for K-Means. In addition to the inefficiency with each MapReduce, for every iteration a new job needs to be started, which means that in addition to the job launching overhead, data needs to be persistent and re-read to/from HDFS. The MPI implementation in contrast loads all points into memory once. For communication the efficient collective layer from MPI (`MPI_Allreduce`) is used.

The Python Scripting implementation (C.1) is based on the Pilot-MapReduce framework, which is an interoperable implementation of the MapReduce for HPC, cloud and Hadoop environments. The framework utilizes Pilots for resource managements. For each map and reduce task, a CU inside the Pilot is spawned. For data-exchange between the tasks the shared filesystem (e.g., Lustre on Stampede) is used. While C.1 outperforms Mahout, it performs significantly worse than MPI, or other hybrid approaches. In particular for larger amounts of shuffle traffic (scenario (ii) and (iii)), Hadoop shuffle implementation is faster. Also, Spark by default compresses the shuffle data, which improves the performance.

Both Spark and HARP are designed to efficiently support iterative workloads such as K-Means. While these approaches cannot entirely reach the performance of MPI, they introduce a unique way of combining the advantages of MPI with Hadoop/MapReduce. Spark performs slightly worse than HARP. However, it must be noted that Spark operates at a higher-level of abstraction, and does not require to operate on low-level data structures and communication primitives. The RDD abstraction provides a consistent key/value-based programming model and provides flexible API for manipulating these. However, since RDD are designed as immutable entities, data often needs to be copied in-memory; in each iteration our K-Means implementation generates two intermediate RDDs. For MPI in contrast only a single copy of the data is stored in memory and manipulated there.

V. DISCUSSION: CONVERGENCE OR INTEGRATION OF HPC WITH HADOOP?

Even though a vibrant ecosystem has evolved to support ABDS objective of providing affordable, scale-out storage and compute on commodity hardware, the increasing processing/-

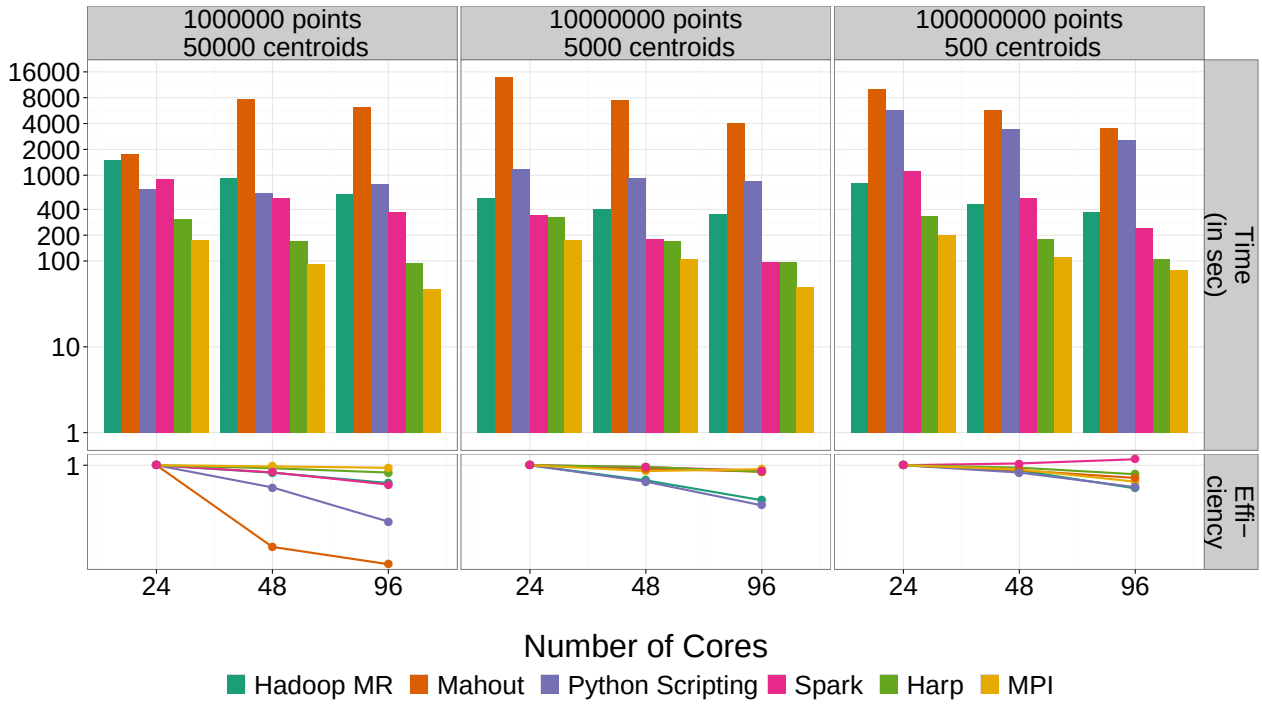

Fig. 2. Runtime of different K-Means Implementations: While MPI clearly outperforms the Hadoop-based implementations, the performance of K-Means can significantly be improved by using hybrid approaches, such as Spark and HARP. By introducing efficient collective and iterative operations known from HPC to Hadoop, the runtime can be improved while maintaining a high-level abstraction for the end-user.

computational requirements, there is a need for convergence between HPC and ABDS ecosystem [55]. Our experiments were designed to expose important distinctions and relevant considerations for integrated ecosystem.

While our micro-benchmark shows that MPI outperforms the Hadoop-based implementation, it must be noted that the second generation Hadoop frameworks, such as Spark, have improved performance significantly by adopting techniques previously only found in HPC, such as effective collective operations. Nonetheless important distinctions remain: Hadoop-based frameworks still maintain a very high and accessible level of abstraction, such as data objects, collections etc., and are typically written without a tight coupling to resource specifics, e.g., the user can modify some parameters, such as the HDFS or RDD chunk size, which also controls the parallelism. In general, frameworks and tools utilize application-level scheduling to manage their workloads and provide powerful abstractions for data processing, analytics and machine learning to the end-user while hiding low-level issues, such resource management, data organization, parallelism, etc. HPC applications operate on low-level, communication operations and application-specific files that often lack a common runtime system for efficiently processing these data objects.

Functionalities available in the ABDS ecosystem (more than 110 implementations) typically exceed those available in the HPC ecosystem, thus reiterating the need for consilience between the two. Several approaches for convergence of the two ecosystems have been proposed. Often, these focus on running Hadoop on top of HPC. However, a lot of the benefits of Hadoop are lost in these approaches, such as data locality aware scheduling, higher cluster utilization etc. Thus, we believe that this is not the right path to interoperability and integration. Furthermore, YARN has been designed to address the needs of data-intensive applications and support application-level scheduling for heterogeneous workloads, there is some ways to go way before YARN can enable both HPC applications and data-intensive applications on the range of resource fabrics found in HPC ecosystem. A possible and promising approach for interoperability that emerges and will be investigated is the extension of HPC Pilot-Job abstraction to YARN, and the usage of Pilot-Data [12] for data-locality aware scheduling.

Our analysis shows that there are technical reasons that drive the convergence between the HPC and ABDS paradigms, e. g. rich and powerful abstractions like collective communications and direct-memory operations, long the staple of HPC are steadily making their presence felt in the ABDS. We anticipate the convergence of conceptual abstractions will soon lead to an integration of tools and technology, e.g., integration of specific capabilities, especially in the form of interoperable libraries built upon a common set of abstractions. In fact, we are working towards such an interoperable library – Scalable Parallel Interoperable Data-Analytics Library (SPIDAL) – that will provide many of the rich data-analytics capabilities of the ABDS ecosystem for use by traditional HPC scientific applications. This will be an incremental but important step towards promoting an integrated approach – the high-performance big-data stack (HPBDS) – that brings the best of both together.

Author Contributions – The experiments were designed primarily by AL and JQ, in consultation with and input from SJ and GCF. The experiments were performed by AL, PM and JQ. Data was analyzed by all. SJ and GCF determined the scope, structure and objective of the paper and wrote the introduction, applications and conclusion. AL wrote the bulk of the remainder of the paper.

Acknowledgement This work is primarily funded by NSF OCI-1253644. This work has also been made possible thanks to computer resources provided by

XRAC award TG-MCB090174 and an Amazon Computing Award to SJ.

REFERENCES

[1] NIST BigData Working Group, http://bigdatawg.nist.gov/usecases.php, 2014.

[2] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002.

[3] A. Sim et al, "GFD.154: The Storage Resource Manager Interface Specification V2.2," Tech. Rep., 2008, oGF.

[4] A. Rajasekar *et al.*, *iRODS Primer: integrated Rule-Oriented Data System*. Morgan and Claypool Publishers, 2010.

[5] S. J. Plimpton and K. D. Devine, "Mapreduce in mpi for large-scale graph algorithms," *Parallel Comput.*, vol. 37, no. 9, pp. 610–632, 2011.

[6] T. Hoefler, A. Lumsdaine, and J. Dongarra, "Towards efficient mapreduce using mpi," in *Proceedings of the 16th European PVM/MPI*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 240–249.

[7] J. Ekanayake *et al.*, "Twister: A runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 810–818.

[8] P. K. Mantha, A. Luckow, and S. Jha, "Pilot-MapReduce: An Extensible and Flexible MapReduce Implementation for Distributed Data," in *Proceedings of third international workshop on MapReduce and its Applications*. New York, NY, USA: ACM, 2012, pp. 17–24.

[9] A. Luckow, M. Santcroos, A. Merzky, O. Weidner, P. Mantha, and S. Jha, "P*: A model of pilot-abstractions," *2012 IEEE 8th International Conference on E-Science*, pp. 1–10, 2012, http://doi.ieeecomputersociety.org/10.1109/eScience.2012.6404423.

[10] I. Raicu, I. T. Foster, and Y. Zhao, "Many-task computing for grids and supercomputers," in *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08) 2008*.

[11] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Gener. Comput. Syst.*, vol. 25, no. 5, pp. 528–540, May 2009.

[12] A. Luckow, M. Santcroos, A. Zebrowski, and S. Jha, "Pilot-Data: An Abstraction for Distributed Data," *Submitted to: Journal of Parallel and Distributed Computing, Special Issue on Big Data*, 2014, http://arxiv.org/abs/1301.6228.

[13] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, Jul. 2002.

[14] A. Tsaregorodtsev *et al.*, "DIRAC3: The new generation of the LHCb grid software," *J.Phys.Conf.Ser.*, vol. 219, p. 062029, 2010.

[15] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 137–150.

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA.

[18] V. K. Vavilapalli, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proc. SOCC*, 2013.

[19] D. Borthakur *et al.*, "Apache hadoop goes realtime at facebook," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11. New York, NY, USA: ACM, 2011, pp. 1071–1080. [Online]. Available: http://doi.acm.org/10.1145/1989323.1989438

[20] M. Zaharia *et al.*, "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*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012.

[21] "Apache TEZ," http://hortonworks.com/hadoop/tez/, 2014.

[22] "Apache Giraph," https://giraph.apache.org/, 2014.

[23] S. Melnik *et al.*, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010.

[24] "Apache Hive," http://hive.apache.org/, 2011.

[25] I. Szegedi, "Pivotal Hadoop Distribution and HAWQ Realtime Query Engine," http://architects.dzone.com/articles/pivotal-hadoop-distribution, 2014.

[26] M. Kornacker and J. Erickson, "Cloudera impala: Real-time queries in apache hadoop, for real," http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/, 2012.

[27] R. S. Xin *et al.*, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD Int. Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 13–24.

[28] A. Abouzeid *et al.*, "Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 922–933, Aug. 2009.

[29] D. J. DeWitt *et al.*, "Split query processing in polybase," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 1255–1266.

[30] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org/

[31] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] "Apache Mahout," http://mahout.apache.org/, 2014.

[33] "RHadoop," https://github.com/RevolutionAnalytics/RHadoop/, 2014.

[34] T. Kraska *et al.*, "Mlbase: A distributed machine-learning system," in *CIDR*. www.cidrdb.org, 2013.

[35] "R on spark," http://amplab-extras.github.io/SparkR-pkg/, 2014.

[36] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Scalable scheduling for sub-second parallel jobs," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-29, Apr 2013.

[37] B. Hindman *et al.*, "Mesos: a platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX, 2011, pp. 22–22.

[38] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 351–364.

[39] Apache Hadoop Project, "Hadoop on demand," http://hadoop.apache.org/docs/r0.18.3/hod.html, 2008.

[40] SAGA, "SAGA-Hadoop," https://github.com/drelu/saga-hadoop, 2014.

[41] myHadoop, "myhadoop," https://portal.futuregrid.org/tutorials/running-hadoop-batch-job-using-myhadoop, 2013.

[42] Amazon Web Services, "Elastic Map Reduce Service," http://aws.amazon.com/de/elasticmapreduce/, 2013.

[43] Microsoft Azure, "HDInsight Service," http://www.windowsazure.com/en-us/services/hdinsight/, 2013.

[44] "Llama," http://cloudera.github.io/llama/, 2013.

[45] W. C. Moody, L. B. Ngo, E. Duffy, and A. Apon, "Jummp: Job uninterrupted maneuverable mapreduce platform," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, 2013, pp. 1–8.

[46] O. Kulkarni, "Hadoop mapreduce over lustre – high performance data division," http://www.opensfs.org/wp-content/uploads/2013/04/LUG2013_Hadoop-Lustre_OmkarKulkarni.pdf, 2013.

[47] P. Zikopoulos *et al.*, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media.

[48] "MapReduce and Lustre: Running Hadoop in a High Performance Computing Environment," https://intel.activeevents.com/sf13/connect/sessionDetail.ww?SESSION_ID=1141, 2013.

[49] Y. Wang *et al.*, "Hadoop acceleration through network levitated merge," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 57:1–57:10.

[50] B. Zhang and J. Qiu, "High performance clustering of social images in a map-collective programming model," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 44:1–44:2.

[51] W. keng Liao, "Parallel k-means data clustering," *http://www.ece.northwestern.edu/~wkliao/Kmeans/index.html*, 2005.

[52] AMP Tutorial, "Spark kmeans," http://ampcamp.berkeley.edu/big-data-mini-course/machine-learning-with-spark.html, 2013.

[53] "XSEDE: Extreme Science and Engineering Discovery Environment," https://www.xsede.org/, 2012.

[54] "Amazon elastic mapreduce," http://aws.amazon.com/elasticmapreduce/.

[55] G Fox, J Qiu and S Jha, High Performance High Functionality Big Data Software Stack, in Big Data and Extreme-scale Computing (BDEC). 2014. Fukuoka, Japan. http://grids.ucs.indiana.edu/ptliupages/publications/HPCandApacheBigDataFinal.pdf.