A Tale of Two Paradigms: Understanding Data-Intensive Approaches using 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, controlling co-placement and scheduling of data with compute resources, and storing, transferring, and managing 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 (HPC) 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" 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 well understood K-means clustering exemplar, and 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. Even though no single application or metric of performance can be a universal determinant of applicability and comparison between the two stacks, we provide a careful generalization of the conclusions from our experiments.**

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 paradigms has been characterized by limited implementations, but customized and tuned for performance along a narrow set of requirements. In contrast, the Apache Hadoop paradigm 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 evolved around a stable, non-monolithic kernel – the Hadoop Filesystem (HDFS).

With the continuing uptake of Hadoop/ABDS, the diversity and heterogeneity of infrastructures will further increase making it increasingly difficult for applications to utilize resources in an interoperable way. The success and evolution of Hadoop/ABDS into a widely deployed cluster computing frameworks yields many opportunities for *traditional* scientific applications; it also raises many important questions, viz., how do typical data-intensive HPC and ABDS workloads differ? what features of the Hadoop/ABDS are useful for traditional scientific workloads? What features of Hadoop/ABDS stack can be extended and integrated with the HPC implementations? Given the drastic and dynamic changes underway, we believe it is premature to attempt closed-form and final answers to these and many other questions. Thus it is the aim of this paper to provide the conceptual framework and terminology to begin addressing these questions.

Paper Outline: This paper can be divided into two logical parts: in the first part, we take a general approach to analyzing the ecosystem of the two prominent approaches to data-intensive applications. We first develop a common basis upon which to analyze the two approaches by proposing common terminology and functional decomposition of the two approaches. which is highlighted as two specific realizations (as shown in Fig 1), hereafter referred to as the high-performance computing (HPC) paradigm and the Hadoop-based big data stack, or for simplicity the Apache stack. We then discuss the salient features of the two stacks, and building upon the underlying terminology compare and contrast the two approaches. In the second part, we move from a fully qualitative exami-

nation of the two stacks, 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. Even though no single application or metric of performance can be a universal determinant of applicability and comparison between the two stacks, we provide a careful generalization of the conclusions from our experiments.

II. Ogres for Big Data

Based upon an analysis of a large set of Big Data applications, including more than 50 use cases [**?**], 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. Note the Big Data Ogres, like the Berkeley Dwarfs are not orthogonal, nor exclusive, and thus do not constitute a formal taxonomy. Also we capture the richness of Big data by including not just different parallel structures (as in 5th ogre below) 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 Ogre captures **different analytical approaches** challenges. Some representative application classes 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: Knowledge discovery often involves fusion of multiple methods (ensemble methods one approach).

The second Ogre captures applications with **important data sources with distinctive features**, representative examples of the data sources include, (i) SQL based, (ii) NOSQL based, (iii) Set of Files (as managed in iRODS), (iv) Internet of Things, (v) Streaming and (vi) HPC simulations.

The third Ogre contains **Distinctive System features**, and includes (i) Agents, as in epidemiology (swarm approaches) and (ii) GIS (Geographical Information Systems).

The forth Ogre builds upon the **Problem Structure** of Big Data applications. For example, (i) Typical N points in a space; important differences between metric and non-metric spaces (ii) Maximum Likelihood, (iii) Chi-squared distributions, and (iv) Expectation Maximization (often method of Steepest descent).

The fifth Ogre organizes **structure of the core analytics kernel** 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), and (xii) Global Optimization (Variational Bayes).

III. Apache Hadoop versus HPC: What can HPC learn from Hadoop?

In this section we give an overview of the HPC and Hadoop ecosystem. We will particularly analyze the different layers and architectural design approaches depicted in Figure 1.

A. High Performance Computing

High Performance Computing (HPC) infrastructure were traditionally built to provide high-end compute infrastructures for scientific applications. HPC is typically 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). The different origins resulted in different software stacks.

Figure 1 compares the HPC and Big Data stack. 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 [8] or GPFS [9] 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 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 are typically shared resources – typically a quota is applied on the data size, but not on I/O. These systems have been particularly optimized with respect to long-running batch jobs. Data locality and other scheduling constraints are not considered.

Lustre and GPFS storage resources are typically exposed as shared filesystem on the compute nodes. In addition several higher-level storage management services, such as SRM [10], iRODS [11] or GFFS, 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. Central to iRODS are the so called micro-services, i. e. the user defined control logic. Micro-services are automatically triggered and handle pre-defined tasks, e. g. the replication of a data set to a set of resources.

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,

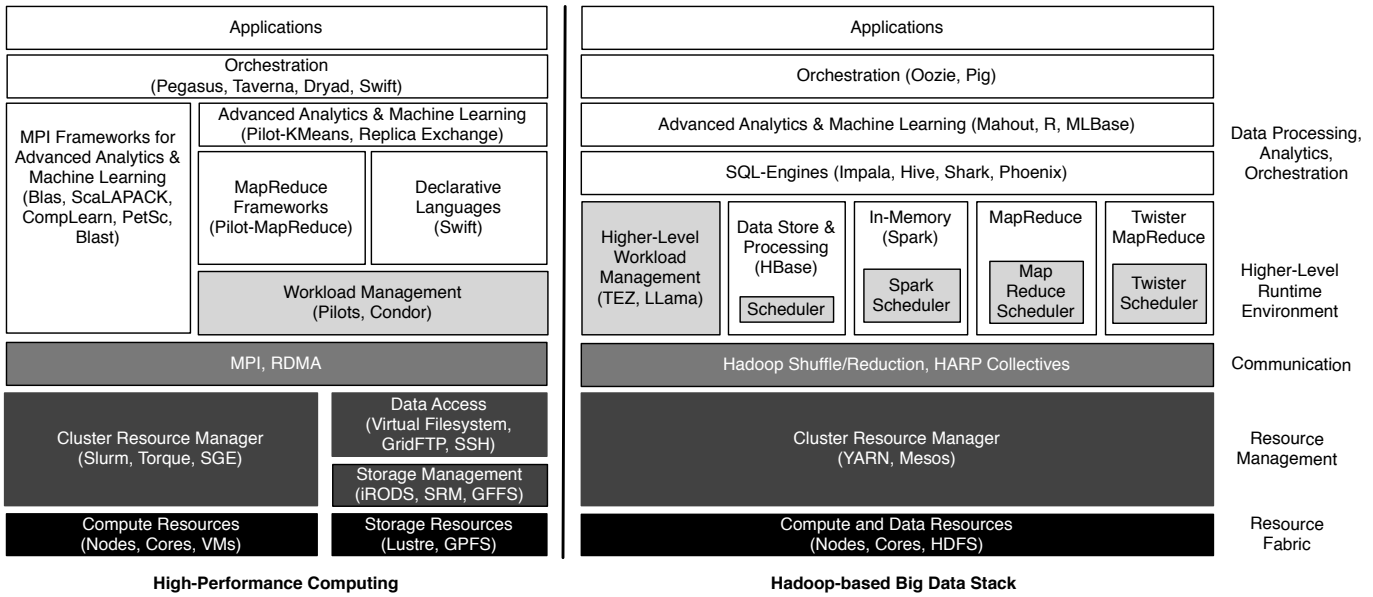

Fig. 1. **HPC and Apache Hadoop Stack:** While HPC infrastructures historically separated data and compute, Apache Hadoop co-locates compute and data. The YARN resource manager heavily utilizes multi-level, data-aware scheduling and provides the basis for a vibrant ecosystem of data processing, analytics and machine learning frameworks. Pilot-Jobs introduced the multi-level scheduling approach to HPC environments and serve as basis for many kinds of workloads.

such as MapReduce-MPI [48], can efficiently utilize HPC features, such as low-latency interconnects and one-sided and non-blocking communications [49]. Further, various non-MPI MapReduce implementations have been proposed: Twister/Salsa [25] to support iterative machine learning workloads, Pilot-MapReduce [50] to support geographically distributed data, etc.

Further, more general approaches for managing larger number of heterogeneous workloads comprising of short-/long-running, data-/non-data intensive, parallel-/non-parallel tasks, have been proposed: Pilot-Jobs [3], many tasks [4], workflows [5] and HTC [6]. Pilot-Jobs e. g. 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 [54], [3]. 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, which provides a basic mechanism for file staging and also supports access to SRM. Another example is Swift [55], which provides a data management component called Collective Data Management (CDM). DIANE provides in-band data transfer functionality over its CORBA channel. DIRAC [56] is an example of a type (ii) system and integrates with SRM storage resources. A disadvantage of these systems compared to the Apache Hadoop stack is while Pilot-Job support dynamic execution, data-management is typically done using files and not using higher level abstractions as found in the Hadoop ecosystem.

B. Hadoop 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 [13]. 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 [**?**] – an open source implementation of the Google Filesystem architecture [**?**] – and the MapReduce framework which was the primary way of processing data stored in HDFS. However, Hadoop saw a broad uptake and the MapReduce model as sole processing model proofed insufficient. Thus, the tight coupling between HDFS, resource management and the MapReduce programming model proofed to be too inflexible for the usage modes that emerged in the Hadoop ecosystem. An examples 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 [35] as central resource manager, Hadoop clusters can now accommodate any application or framework. Figure 1 (right) depicts a Big Data stack based on Hadoop 2's HDFS and YARN. Based on the resource fabric and management layer a vibrant ecosystem of higher-level runtime systems, data processing and machine learning libraries emerged. Historically,

MapReduce was the Hadoop runtime layer for processing data; but, meet the application requirements runtimes for record-based, random-access data (HBase [**?**]), iterative processing (Spark [15], TEZ [37], Twister [25]), stream (Spark Streaming) and graph processing (Apache Giraph [**?**]) emerged. A key enable for these frameworks is the YARN support for multi-level scheduling, which enables the application to deploy there 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 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 e. g. 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. As shown in section IV Spark provides a powerful abstraction for expression iterative algorithms, such as KMeans while showing a good performance compared to earlier approaches such as Apache Mahout [22].

Many analytical applications – in particular in enterprise environments – rely on SQL as primary data query language. This lead to the development of several SQL-based analytic engines: Google's Dremel [16], Hive [17], HAWQ [**?**], Impala [18] and Shark [19] are prominent examples of such SQL 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 [**?**] e. g. deploys a PostgreSQL database on every node on which it distributes the data using hash partition. Further, the system can access data from HDFS via external tables. Further, several commercial approaches with a similar exist, e. g. Polybase [**?**].

Higher up in the stack are machine learning applications; in contrast to typical SQL-queries, these typically require multiple iterations on the data. While traditional in-memory, single-node tools, such R [20] or Scikit-Learn[21] provide a powerful implementations of many machine learning algorithms, they are mostly constrained to a single machine. To overcome this limitation, several approaches for using Hadoop/MapReduce for ML implementations have been proposed, e. g. Apache Mahout [22] or RHadoop [24]. There are several, well-known limitations of Hadoop 1 with respect to support for iterative MapReduce applications [25]. Thus, increasingly, these iterative runtimes are deployed. MLBase [26] is a machine learning framework based on Spark as lower-level data processing framework. SparkR [**?**] allows R applications to utilize Spark.

C. Hadoop Resource Management

Historically, many scientific applications comprises of large, monolithic simulations spawning a large number of cores running on High Performance Computing (HPC) infrastructures. These applications typically utilize fine-grained, tightly-coupled parallelism (often based MPI). Such applications typically allocate a static space of the HPC system and need to be scheduled in a way that they simultaneously execute on a system they, e. g. using Gang scheduling. Typically workloads in such environments are rigid: MPI jobs e. g. comprise of a set of tightly coupled highly latency sensitive tasks with fixed resource demands, which do not change during a jobs lifetime. Typically, the amount of resources and walltime is defined at submission time and does not change during runtime. Data locality is not a first order concern: most HPC are write heavy, while data-intensive applications (e. g. analytics applications) are more read heavy.

Data-intensive workloads can easily be decomposed into multiple, very fine-grained parallel tasks. One hypothesis is that by using many small tasks the utilization of resources and fairness can be improved [38], [28]. Also, this type of fine-grained parallelism forms the basis for supporting interactive workloads on top of distributed infrastructure. Scheduling heterogeneous workload consisting of small, short-running tasks and longer running more batch oriented tasks represents a challenge for traditional monolithic, centralized cluster scheduling systems. In addition the resources dynamisms introduced by the multi-tenant nature of such cluster systems increases the difficulty.

Different architectures for resource management in HPC and Apache Hadoop environments have been proposed: centralized, multi-level and decentralized. Table I summarizes different architectures. HPC schedulers were designed for very rigid applications and frameworks with constant resource requirements during their runtime, such as parallel applications based on MPI. Multi-level schedulers (such as Mesos [7]) emerged to address the need to efficiently support data-intensive workloads and allow the dynamic allocation and usage of resource through application-level scheduling. To overcome scalability bottlenecks in the centralized and even multi-level approach when running a large number of short running tasks, some decentral scheduler architectures have been proposed, e. g. Google's Omega [27] and Sparrow [28], an application-level scheduler of Spark.

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 to resource and cluster management service significantly constraining flexibility. However, in particular in the early days this was not an issue: Hadoop was often used on traditional HPC clusters using Hadoop on Demand [29], SAGA Hadoop [30] or MyHadoop [31], the native Hadoop support in schedulers such as SLURM or in clouds (Amazon's Elastic MapReduce [32] or Microsofts

	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	High (Application-Level Scheduling)	High (Application-Level Scheduling)

TABLE I
SCHEDULER ARCHITECTURES: INCREASINGLY, CENTRALIZED AND MONOLITHIC SCHEDULING SYSTEMS ARE BEING REPLACED TO SUPPORT HIGH-VOLUME, SHORT-RUNNING TASKS IN COMPUTE/DATA CLUSTER RESOURCES.

HDInsight [33] on Azure). A main limitation of these approaches is the fact that data has to be initially moved to the HDFS filesystem before running the computation, i. e. data locality can generally not exploited.

Despite the limitations of Hadoop 1, many different resources usage modes for Hadoop clusters emerged. Historically, 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 [35], [34] the core of Hadoop 2 was designed to address the need to efficiently support heterogeneous workloads in larger Hadoop cluster environments. One design objective is to provide effective support for the fine-grained task-level parallelism exhibited by data-intensive applications. With the increasing size of these Hadoop cluster and the greater variety of Hadoop-based 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. While YARN solves some of these problems, it has some limitations: it provides e. g. only a very low-level abstraction for resource management; data locality needs to be manually managed by the application by requesting resources at the location of an file chunk. Mesos [7]; Mesos is a two-level scheduler that tightly integrates application and resource management system.

YARN provides the basis for application-level scheduling. As shown in Figure 1 applications frameworks typically rely on a runtime system that embeds an application-level scheduler; this is e. g. the case for MapReduce, Spark and HBase. Another observation is the fact that increasingly the common application runtime requirements are integrated into higher-level runtime systems frameworks, e. g. the support for long-running applications or multi-stage applications using DAGs (directed acyclic graph). For example, Llama [36] offers a long-running application master for YARN application designed for the Impala SQL engine. Similar capabilities for Mesos are provided by Chronos [40] and Marathon [41]. In the HPC world similar capabilities exists via Pilot-Jobs [3]. TEZ [37] is a DAG processing engine primarily designed to support the Hive SQL engine.

YARN and Mesos expose a much more dynamic resource model to the application than typical HPC schedulers. Generally, these workloads consisting of short-running tasks; this enables the scheduler e. g. to easily remove resources from an application (by simply waiting until task completion). Thus, the overall system can better exploit a dynamically expanding or shrinking resource pool and improve the overall cluster utilization. For short-running jobs the scheduler often represents a bottleneck, e. g. the startup of a YARN application typically requires several seconds. However, to enable this form of dynamic resource usage, both Mesos or YARN require a tighter integration of the application and the resource management system. In YARN, the application needs to register a so called Application Master with YARN. As part of this process the Application Master subscribes to a set of callbacks. The unit of scheduling is referred to as a container. Containers are requested from the so called Resource Manager. The manager does not necessarily returns the requested number of resources, i. e. the application is required to elastically utilize the assigned resources; Also, YARN can request the de-allocation of containers. Thus, the application needs to carefully keep track of resources allocated and de-allocated. Similarly, in Mesos e. g. the application-level scheduler is required to register with the Mesos master and needs to respond to various callbacks, such as resource offers and/or revocations.

D. High-Performance Big Data System (HPBDS): 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 vs. cheap data storage and retrieval in the Hadoop case, a convergence of both worlds 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. The Hadoop ecosystem matured to support a wide range of heterogeneous workloads particularly with the introduction of YARN and Mesos. 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.

A particular advantage of Hadoop is that the ecosystem provides a wide-range of higher-level abstraction for data storage and processing/analytics (MapReduce, iterative MapReduce, graph analytics, machine learning etc.) all built on top of an extensible kernel, the Hadoop Filesystem and YARN. In contrast to HPC schedulers, YARN has first-order design objective 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. For MapReduce e. g. an application can request a different

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
KMEANS – COMPARISON OF DIFFERENT ARCHITECTURES AND APPROACHES

number of slots for the map and the reduce phase.

While HPC typically do not deploy YARN as primary scheduler, the uptake of Hadoop has lead to various proposals for integrating Hadoop/YARN into HPC environments. HPC environments typically decouple compute and storage and connect both with high-bandwidth, low-latency networks. The following integration aspects need to be addressed: (i) the integration with the local resource management level system, (ii) the integration with HPC storage resources (i. e. the shared, parallel filesystem) and (iii) the usage of high-end network features such as RDMA as well as the implementation of efficient abstractions (e. g. collective operations) on top of these.

Resource Management Integration: To achieve integration with the native, system-level resource management system (i), 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 [30], JUMMP [42] or MyHadoop [31], 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 typically deploy 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 anywhere Posix compliant filesystem. Thus, most parallel filesystems can easily be used in conjunction with Hadoop (ii)- however, in these cases the Hadoop layer will not be aware of the data locality maintained on the parallel filesystem level. Intel e. g. supports Hadoop on top of Lustre [43], IBM on top of GPFS [44]. Another optimization concerns the MapReduce shuffling phase that is carried out via the shared filesystem [45]. Also, the scalability of these filesystem is usually constraint compared 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 (iii) 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 [46], [47]. HARP introduces an abstraction for collective operations with Hadoop jobs [65].

IV. EXPERIMENTS

In the following we run different KMeans implementation on different kinds of infrastructures. Table II summarizes the different implementations of the KMeans algorithm. 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 [22], for (B) an MPI-based KMeans implementation [63]. Further, we examine the following hybrid approaches: (C.1) Python Scripting implementation using Pilots [50] (Pilot-KMeans), (C.2) a Spark KMeans [64] and (C.3) a HARP implementation [65]. While (C.1) provides an interoperable implementation of the MapReduce programming model particularly for HPC environments, (C.2) and (C.3) enhance Hadoop for efficient iterative computations and introduce collective operations to Hadoop environments.

We use three different infrastructures: Amazon EC2, the Madrid YARN/Hadoop cluster and the Stampede clusters (which is part of XSEDE [**?**]). On EC2 we utilize the cluster compute instance type, which provides a HPC-style environment. We utilize Elastic MapReduce [**?**] for managing the Hadoop cluster in scenario (A.1) and the `spark-ec2` tool for scenario (C.3). Madrid uses YARN as resource manager, Stampede is deploying SLURM.

We run three different KMeans 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 KMeans iterations 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 defined by the number of points $\times$ number of clusters, the amount of data that needs to be exchanged during the shuffle phase increases gradually from scenario (i) to (iii) proportionally to the number of points.

Figure 2 shows the results of this experiment. Both Hadoop implementations of KMeans (Hadoop MR (A.1)/Mahout (A.2)) perform significantly worse than the HPC implementation based on MPI. The Map Reduce model – the predominant usage mode of Hadoop 1.0 – 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 particular in case of larger amounts of shuffle data, data needs to be spilled to disks. For KMeans for e. g. a sorting is not essential. 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 persisted and re-read to/from HDFS. The MPI implementation in contrast only loads all points into memory once. For communication the efficient collective layer from MPI (`MPI_Allreduce`) is used.

The Python Scripting implementation (Pilot-KMeans) (C.1) is based on the Pilot-MapReduce framework, which is an interoperable implementation of the MapReduce programming model 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 (the Lustre scratch directory on Stampede) is used. While the Python implementation outperforms Mahout, it performs significantly worse than MPI or other hybrid approaches. In particular for larger amounts of shuffle traffic (scenario (ii) and (iii)), the 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 KMeans. 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 on a higher-level of abstraction and do 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 KMeans implementation generates two intermediate RDDs. For MPI in contrast only a single copy of the data is stored in memory and manipulated there.

V. Conclusion and Future Work

A vibrant ecosystem of framework and tools evolved on top of the Hadoop Filesystem and YARN. While the primary design objective of Apache Hadoop was affordable, scale-out storage/compute on commodity hardware, with the increasing compute requirements both the HPC and Hadoop stack converge. Compute-intensive, parallel workloads on both HPC and Hadoop significantly benefit from high-end CPUs, memory and interconnects. Also, there is a potential role for accelerators (e. g. Nvidia GPGPUs or Intel MICs) in the Hadoop stack.

YARN enables users to run HPC-style applications e. g. based on MPI, as well as data-intensive applications on Hadoop clusters. However, YARN has been primarily designed to address data-intensive requirements, such as data locality, fine-grained parallelism with many short-running compute units and support for heterogeneous workloads. Hadoop applications are typically written on a higher level of abstraction (MapReduce, SQL, Spark) without resource specifics in mind. If desired, the user can modify some parameters, e. g. the HDFS or RDD chunk size, which also controls the parallelisms. MPI applications in contrast operate on low-level data types and communication operations.

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, significantly improved performance by adopting technique previously only found in HPC, such as effective collective operations. At the same time these frameworks still maintaining a very high and accessible level of abstraction, such as data objects, collections etc. HPC applications operate on low-level, application-specific files that often lack a common runtime system for efficiently processing these data objects. The Apache stack demonstrates the power of application-level scheduling that lead to the development of many vertical frameworks that 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. The functionalities available in the Apache Hadoop stack clearly exceeds the ones available in the HPC stack.

Several approaches for combining HPC and Hadoop have been proposed. Often, these focus on running Hadoop on top of a standard HPC stack consisting of a resource manager as SLURM and a parallel filesystem as Lustre. 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. A possible approach for interoperability that we will investigate in the future are Pilot-Jobs. By extending HPC Pilot-Jobs to YARN and the usage of Pilot-Data [54] for data locality aware scheduling a promising approach for combining both worlds can be achieved.

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

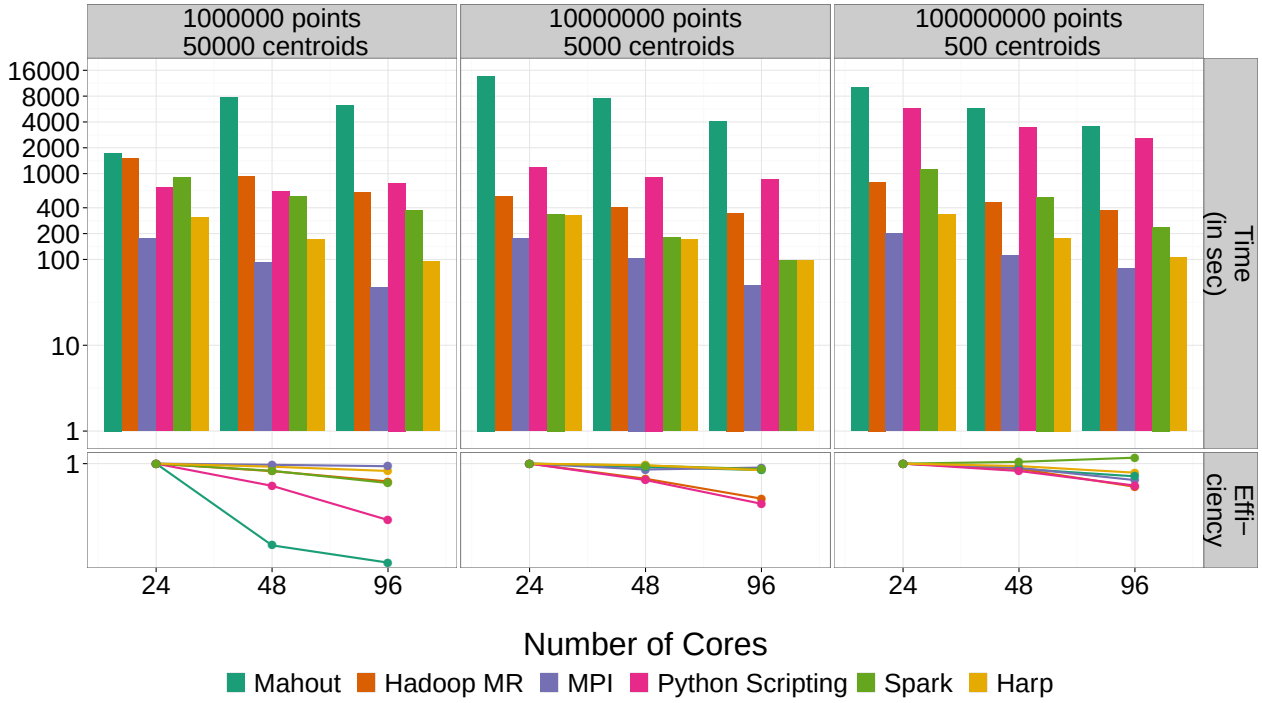

Fig. 2. Runtime of different KMeans Implementations

determined the scope, structure and objective of the paper and wrote the introduction. AL wrote the bulk of the remainder of the paper.

Acknowledgement This work is primarily funded by NSF CAREER Award (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] D. G. Feitelson and L. Rudolph, "Gang scheduling performance benefits for fine-grain synchronization," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 306–318, 1992.

[2] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington: Microsoft Research, 2009.

[3] A. Luckow, M. Santcroos, O. Weidner, A. Merzky, P. Mantha, and S. Jha, "P*: A Model of Pilot-Abstractions," in *8th IEEE International Conference on e-Science 2012*, 2012.

[4] 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*.

[5] 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. [Online]. Available: http://dx.doi.org/10.1016/j.future.2008.06.012

[6] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for high throughput computing," *SPEEDUP Journal*, vol. 11, no. 1, June 1997.

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "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 Association, 2011, pp. 22–22. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[8] Oracle, "Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System (White Paper)," 2007.

[9] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083323.1083349

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

[11] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, *iRODS Primer: integrated Rule-Oriented Data System*. Morgan and Claypool Publishers, 2010.

[12] S. Jha, M. Cole, D. S. Katz, M. Parashar, O. Rana, and J. Weissman, "Distributed computing practice for large-scale science and engineering applications," *Concurrency and Computation: Practice and Experience*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.2897

[13] 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.

[14] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer, "Beowulf: A parallel workstation for scientific computation," in *In Proceedings of the 24th International Conference on Parallel Processing*. CRC Press, 1995, pp. 11–14.

[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[16] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm

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

[18] 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.

[19] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*,

ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465288
[20] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org/
[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[22] "Apache Mahout," http://mahout.apache.org/, 2014.
[23] "Oryx," https://github.com/cloudera/oryx, 2014.
[24] "Rhadoop," https://github.com/RevolutionAnalytics/RHadoop/wiki, 2014.
[25] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "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. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851593
[26] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "Mlbase: A distributed machine-learning system," in *CIDR*. www.cidrdb.org, 2013.
[27] 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. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465386
[28] 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. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-29.html
[29] Apache Hadoop Project, "Hadoop on demand," http://hadoop.apache.org/docs/r0.18.3/hod.html, 2008.
[30] SAGA, "SAGA-Hadoop," https://github.com/drelu/saga-hadoop, 2014.
[31] myHadoop, "myhadoop," https://portal.futuregrid.org/tutorials/running-hadoop-batch-job-using-myhadoop, 2013.
[32] Amazon Web Services, "Elastic Map Reduce Service," http://aws.amazon.com/de/elasticmapreduce/, 2013.
[33] Microsoft Azure, "HDInsight Service," http://www.windowsazure.com/en-us/services/hdinsight/, 2013.
[34] V. K. Vavilapalli, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proc. SOCC*, 2013.
[35] "Apache Hadoop NextGen MapReduce (YARN)," http://hadoop.apache.org/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/YARN.html, 2013.
[36] "Llama – low latency application master," http://cloudera.github.io/llama/, 2013.
[37] "Apache tez," http://hortonworks.com/hadoop/tez/, 2013.
[38] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica, "The case for tiny tasks in compute clusters," in *Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems*, ser. HotOS'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2490483.2490497
[39] M. Rynge, G. Juve, G. Mehta, E. Deelman, K. Larson, B. Holzman, I. Sfiligoi, F. Wurthwein, G. B. Berriman, and S. Callaghan, "Experiences using glideinwms and the corral frontend across cyberinfrastructures," in *Proceedings of the 2011 IEEE Seventh International Conference on eScience*, ser. ESCIENCE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 311–318. [Online]. Available: http://dx.doi.org/10.1109/eScience.2011.50
[40] "Chronos," http://airbnb.github.io/chronos/, 2014.
[41] "Marathon," https://github.com/mesosphere/marathon, 2014.
[42] 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.
[43] 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.
[44] IBM, P. Zikopoulos, and C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1st ed. McGraw-Hill Osborne Media, 2011.
[45] "Mapreduce and lustre: Running hadoop* in a high performance computing environment," https://intel.activeevents.com/sf13/connect/fileDownload/session/656FF9F9557ADC9ACFAE08B4B2F3831B/SF13_SFTS010_100.pdf, 2013.
[46] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal, "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. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063461
[47] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, "High performance rdma-based design of hdfs over infiniband," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 35:1–35:35. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389044
[48] S. J. Plimpton and K. D. Devine, "Mapreduce in mpi for large-scale graph algorithms," *Parallel Comput.*, vol. 37, no. 9, pp. 610–632, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2011.02.004
[49] T. Hoefler, A. Lumsdaine, and J. Dongarra, "Towards efficient mapreduce using mpi," in *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 240–249. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03770-2_30
[50] 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*, ser. MapReduce '12. New York, NY, USA: ACM, 2012, pp. 17–24.
[51] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: A Fast and Light-Weight TasK ExecutiON Framework," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1–12.
[52] I. Raicu, Y. Zhao, I. T. Foster, and A. Szalay, "Accelerating large-scale data exploration through data diffusion," in *Proceedings of the 2008 international workshop on Data-aware distributed computing*, ser. DADC '08. New York, NY, USA: ACM, 2008, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/1383519.1383521
[53] Z. Zhang, D. Katz, J. Wozniak, A. Espinosa, and I. Foster, "Design and analysis of data management in scalable parallel scripting," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, 2012, pp. 1–11.
[54] 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.
[55] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.
[56] A. Tsaregorodtsev, N. Brook, A. Casajus Ramo, P. Charpentier, J. Closier *et al.*, "DIRAC3: The new generation of the LHCb grid software," *J.Phys.Conf.Ser.*, vol. 219, p. 062029, 2010.
[57] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, A. Hayrapetyan, A. Harutyunyan, A. J. Peters, and P. Saiz, "Alien: Alice environment on the grid," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062012, 2008. [Online]. Available: http://stacks.iop.org/1742-6596/119/i=6/a=062012
[58] T. Maeno, K. De, T. Wenaus, P. Nilsson, G. A. Stewart, R. Walker, A. Stradling, J. Caballero, M. Potekhin, D. Smith, and T. A. Collaboration, "Overview of atlas panda workload management," *Journal of Physics: Conference Series*, vol. 331, no. 7, p. 072024, 2011. [Online]. Available: http://stacks.iop.org/1742-6596/331/i=7/a=072024
[59] T. Maeno, K. De, and S. Panitkin, "PD2P: PanDA dynamic data placement for ATLAS," in *Journal of Physics: Conference Series*, vol. 396, 2012, p. 032070. [Online]. Available: http://iopscience.iop.org/1742-6596/396/3/032070
[60] Continuuity, "Weave," https://github.com/continuuity/weave, 2014.
[61] Cloudera, "Kitten," https://github.com/cloudera/kitten, 2014.
[62] A. Luckow, L. Lacinski, and S. Jha, "SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems," in *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010, pp. 135–144.

[63] W. keng Liao, "Parallel k-means data clustering," *http://www.ece.northwestern.edu/~wkliao/Kmeans/index.html*, 2005.
[64] A. C. Tutorial, "Spark kmeans implementation," 2013.
[65] 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. [Online]. Available: http://doi.acm.org/10.1145/2523616.2525952