Big Data Analytics on HPC Architectures: Performance and Cost

Peter Xenopoulos[1], Jamison Daniel[2], Michael Matheson[2] and Sreenivas Sukumar[2]
peter.xenopoulos@pomona.edu, danieljr@ornl.gov, mathesonma@ornl.gov, sukumarsr@ornl.gov

***Abstract*— Data driven science, accompanied by the explosion of petabytes of data, has called into need dedicated analytics computing resources. Dedicated analytics clusters require large capital outlays due to their expensive hardware requirements. Additionally, if such resources are located far from the data they analyze, they also incur substantial data transfer, which has both cost and latency implications. In this paper, we benchmark a variety of high-performance computing (HPC) architectures for classic data science algorithms, as well as conduct a cost analysis of these architectures. Additionally, we compare algorithms across analytic frameworks, as well as explore hidden costs in the form of queuing mechanisms. We observe that node architectures with large memory and high memory bandwidth are better suited for big data analytics on HPC hardware. We also conclude that cloud computing is more cost effective for small or experimental data workloads, but HPC is more cost effective at scale. Additionally, we quantify the hidden costs of queuing and how it relates to data science workloads. Finally, we observe that software developed for the cloud, such as Spark, performs significantly worse than pbdR when run in HPC environments.**

I. Introduction

High-performance computing (HPC) architectures play a vital role in computational science, especially in simulations with applications in quantum mechanics, climate science and combustion. Typically, these architectures can take two roles: *capacity* machines, which provide multiple users access to compute resources to solve many small problems, or *capability* machines, which solve a single large problem using a high fraction of the machine's capacity. A unique case of a capability machine is the 18,688-node Titan supercomputer, part of the Oak Ridge Leadership Computing Facility (OLCF) at Oak Ridge National Laboratory (ORNL) [1]. Capability type jobs include problems such as climate or combustion simulations. Examples of capacity machines include computing clusters at small research institutions and businesses. Problems that fit onto capacity machines may be small and quick data analytics workloads. Both types of machines are vital tools for scientists and have not only allowed scientific institutions to increase scientific output, but also to solve problems previously deemed unsolvable. For these reasons, many institutions house HPC resources.

The new trend of data driven science accompanied by the explosion of data from atypical sources, such as sensors and social media, brings to the forefront a need for dedicated analytics computing clusters [2], [3]. Typical analytics clusters require large capital investment due to their expensive hardware requirements, including terabytes of RAM, graphical processing units (GPUs) and lightning-fast network connections. Additionally, dedicated analytics resources may not be local to the data, potentially even thousands of miles away, as is the case for cloud resources. This can incur high amounts of data transfer, which has both cost and latency implications [4]. Finally, typical HPC architectures may also impose strict queuing policies, which inhibit the time sensitive nature of analytics. For many institutions, seeking the best performance in computing resources at the lowest cost is of paramount importance for budgetary reasons. With our study, we seek to uncover the cost of performance for data analytics.

Most benchmarking studies focus solely on the computing performance of computational clusters, and as such, do not attribute a cost to the performance measured [5], [6]. Previous cost-benefit work has focused on cloud and volunteer computing, but not on comparing HPC architectures [7]. However, there has been research done in benchmarking the performance of scientific applications in the cloud [8]. Specifically, the results concluded that for data-intensive applications, the storage costs were insignificant compared to compute costs. The study, however, did not explore the cost analysis compared to traditional HPC alternatives to scientific applications. Further studies into the use of cloud storage for scientific applications showed that systems such as Amazon S3 were unsuitable for scientific applications [9]. Performance benchmarking studies of cloud computing services such as Amazon EC2 concluded that cloud computing, particularly for MPI based applications, was still far off from competing with traditional HPC architectures [10]. Although comparisons and cost analysis between traditional HPC and cloud computing are few and far between, previous work has indeed found that for research institutions like a university, operating a community cluster program is more cost effective than resorting to a cloud computing service [11]. Our study is different from previous studies in that we benchmark various

*This manuscript has been co-authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doepublic-access-plan). This project was partially funded by the Laboratory Director's Research and Development fund. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy.

[1]P. Xenopoulos is part of Pomona College, Claremont, CA 91711

[2]J. Daniel, M. Matheson and S. Sukumar are part of the Advanced Data and Workflows Group at the National Center for Computational Sciences in Oak Ridge National Laboratory, Oak Ridge, TN 37831

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

types of leadership computing hardware particularly for data science workloads.

With our study, we explore the performance of traditional data science algorithms across architectures and analytical frameworks as well as model the cost of each. We specifically explore the performance characteristics of matrix multiplication, singular value decomposition (SVD), linear regression and k-means. We benchmark each hardware based on these four algorithms for a variety of data sizes. Our findings can be summarized as follows: (i) nodes with lots of memory as well as high memory bandwidth are better suited for big data analytics on HPC hardware, (ii) small or experimental data workloads are better off on cloud computing architectures due to their flexibility and availability, (iii) as data workloads become larger, cost at scale favors HPC, (iv) there is a hidden cost of waiting on traditional HPC architectures in the form of a queuing mechanism, and finally (v) software developed for the cloud, such as Spark, performs worse than its MPI comparables when run in HPC environments.

Our paper is structured in the following order. In section II, we discuss our hardware used, data science frameworks, algorithms benchmarked and cost estimate procedure. In section III, we discuss our benchmark performance results. In section IV, we discuss the effects of memory bandwidth and queue systems on performance and conduct a cost benefit analysis of HPC and cloud computing. Finally, in section V, we conclude and make suggestions to various data science users.

II. Methods

A. Hardware

The hardware used in the benchmarks consists of three HPC resources part of the OLCF at ORNL [12]. The chosen assortment of HPC hardware gives a broad range of HPC hardware (at the node level) available at scientific institutions. The hardware of each resource is contained in Figure 1.

Each of these resources is local to a shared Lustre filesystem and has no node storage. The Titan supercomputer contains 18,688 compute nodes and is used primarily for simulation tasks. Rhea is a commodity Linux cluster used for data pre- and post-processing, with 512 non-GPU compute nodes and 9 GPU compute nodes. Eos is a general purpose 736 node Cray XC30 cluster. More detailed information about each resource is shown in Figure 1 and Table I. The memory bandwidth benchmarks were conducting using the STREAM benchmark library [13].

B. Analytics Frameworks

We conducted benchmarks on each architecture using two analytical frameworks, Apache Spark and pbdR. The selection of these two frameworks is predicated on their popularity among many researchers due to their speed and distributed approach.

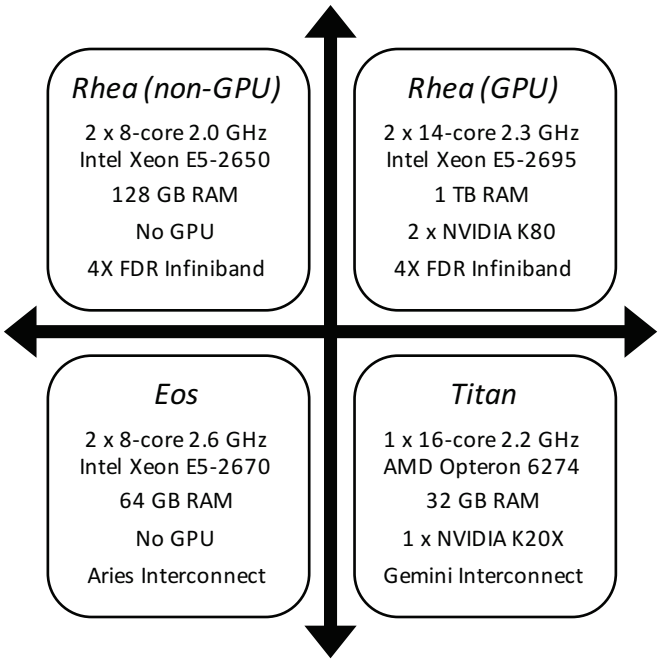

Fig. 1. OLCF node level hardware

1) Spark: Spark is an open-source computing framework currently maintained by the Apache Foundation. Spark addresses the limitations of the MapReduce framework's to effectively deal with iterative algorithms. Instead of typical MapReduce operations such as reading and writing data to disk, Spark utilizes a resilient-distributed dataset (RDD) and tries to keep in memory as much of the data as possible [14]. RDDs are an immutable data structure that are distributed over a cluster and are fault-tolerant. The use of RDDs facilitates efficient computation of iterative algorithms, as well as interactive exploratory data analysis, which contributes to Spark's popularity. Spark's machine learning capabilities are contained in a library called MLlib [15]. Typically, we see Spark used on inexpensive commodity clusters, like Rhea, because of its RDD data structure, which make its applications fault tolerant. For our purposes, Rhea (non-GPU) is the most similar hardware to what Spark would usually run on. Additionally, Spark is seeing more use on cloud computing architectures.

2) pbdR: pbdR is a collection of R packages which utilizes a single program multiple data paradigm to provide an environment for statistical computing and big data [16]. pbdR focuses on distributed memory systems, such as the HPC clusters listed earlier, and does not focus on interactive computing nor on master/workers systems such as Spark. In pbdR, communications between processors is based on the Messaging Passing Interface (MPI). Part of the collection of packages in pbdR are the pbdDMAT, pbdML and pmclust packages, which provide extensions for distributed matrix operations and machine learning algorithms [17], [18]. pbdR was developed for use on HPC architectures, specifically those found at ORNL, so we would expect it to perform quite well on Titan and Eos.

TABLE I
OLCF NODE LEVEL HARDWARE CHARACTERISTICS

	L2 Cache	L3 Cache	RAM
Rhea	16 x 256 KB	40 MB	DDR3
Rhea (GPU)	26 x 256 KB 8-way set associative caches	70 MB 20-way set associative shared cache	DDR4
Titan	8 x 2 MB 16-way set associative shared exclusive caches	2 x 8 MB up to 64-way set associative shared caches	DDR3
Eos	16 x 256 KB 8-way set associative caches	40 MB 20-way set associative shared cache	DDR3

C. Algorithms

To conduct our benchmarks, we selected four algorithms common to data analytics applications: matrix multiplication, singular value decomposition (SVD), linear regression and k-means. These algorithms represent a wide array of not only matrix operations but also iterative optimization problems commonly observed in data science.

1) Matrix Multiplication: Matrix multiplication is a worthwhile benchmark because it is the foundation of many data science applications. Let A and B both be $n \times n$ matrices. In order to calculate $A \times B = AB$, we need to perform at least $2n^3 - n^2$ operations. In a sequential algorithm, this can take a lot of time for even seemingly small matrices, so it makes sense to parallelize matrix multiplication, which is considered an "embarrassingly parallel" algorithm [19].

2) Singular Value Decomposition: The SVD is a matrix factorization which has many applications in signal processing and statistics and is used commonly by data scientists. Let B be an $m \times n$ matrix. Through SVD, we can decompose B into:

$$B = U\Sigma V^T \tag{1}$$

where U is an $m \times m$ unitary matrix, Σ is an $m \times n$ matrix with non-negative real numbers on the diagonal, and V^T is an $n \times n$ unitary matrix. The entries, $\sigma_{i,i}$ of Σ contain the singular values of B. The pbdR package uses a probabilistic algorithm to construct an approximate singular value decomposition [20].

3) Linear Regression: Linear regression is a common machine learning algorithm. Given a data set $\{y_i, x_{i1}, x_{i2}, ..., x_{ip}\}_{i=1}^{n}$, where n is the number of samples and p is the number of predictors, we can model the relationship between x and y as

$$Y = X\beta \tag{2}$$

where $Y = (y_1, y_2, ..., y_n)^T$, $X = (x_1, x_2, ..., x_p)$, where x_i is a column vector of length n and β is the p-dimensional parameter vector to be estimated. pbdR uses fits a linear model by using a QR decomposition with a limited pivoting strategy.

4) k-means: k-means is a popular clustering algorithm. Given a vector of observations $\vec{x} = (x_1, x_2, ..., x_n)$ where each observation is a d-dimensional vector, k-means partitions n observations into k sets $S = S_1, S_2, ..., S_k$. In doing so, k-means calculates:

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{3}$$

where μ_i is the centroid of S_i.

D. Experimental Design

We tested each algorithm across Titan, Eos and both the Rhea GPU and non-GPU nodes. We tested each algorithm on 1, 2 and 5 node jobs. By testing on various node sizes of different architectures, we are able to test both the scale-up and scale out properties of each algorithm. For matrix multiplication and SVD, we used several $n \times n$ matrices. For linear regression and k-means, we used $n \times 2$ matrices. This means, for linear regression, we regressed one independent variable, x on one dependent variable, y. For k-means, we clustered the two-dimensional data into $k = 4$ partitions. For matrix multiplication, we used $10,000 \times 10,000$, $20,000 \times 20,000$ and $30,000 \times 30,000$ matrices. For SVD, we used the same matrices, plus a $40,000 \times 40,000$ matrix, due to differing memory requirements between SVD and matrix multiplication. For linear regression and k-means, we used 1, 5, 10, 25 and 50 million observations. Each matrix was dense and had entries that were distributed $N(0, 1)$. We used dense matrices because they represent the "worst case scenario". We used Hierarchical Data Format, HDF, to compress the matrices so that they would lower Input/Output (I/O) time and represent how scientific users store files of such sizes [21].

In our experiment, we treated the Rhea (GPU) nodes as our "analytics" nodes. These nodes are considered "fat" nodes, due to their terabyte of memory and two NVIDIA accelerators per node. Such characteristics may be useful for data analytics and iterative algorithms, as one would have to request fewer nodes, because each node could hold more data in memory. This would also reduce communications between nodes in a single job. For cloud computing, where the interconnect is slower than in traditional HPC infrastructures, such structure make a significant difference [22]. The Rhea (non-GPU) nodes, called "skinny" nodes, represent more classic cloud-computing architecture. Finally, Titan and Eos represent classic HPC architectures typically used for simulation purposes. They feature a strong interconnect, however lack the RAM size that both the Rhea fat and skinny nodes possess. For data science problems of large size, this will drive up the fixed cost to perform data science, as more nodes will have to be requested to complete a certain workload. Additionally, there may be increased communications costs between nodes as data has to be partitioned among more nodes.

For each test of an algorithm, we recorded the time for five processes: Initialization, I/O, Blockcyclic, Computation and Finalize. Initialization is a relatively short process which sets up Message Passing Interface (MPI) communicators between

nodes. This was fairly constant between all machines. I/O is the time taken for the data to be read and stored in memory. Blockcyclic is the time taken to distribute the matrix effectively across the nodes in blocks. Computation is the time taken for the algorithm to run. Finalize is the time taken for MPI communicators to terminate.

E. Deriving Cost

To develop a cost estimate for the performance on each machine, we took into account both the fixed and variable cost, denoted as F and V respectively. We used a simple cost formula where cost depends on the machine and time used, t. The total cost, T, of using resource A for time t would be:

$$T_A = F_A + V_A(t) \tag{4}$$

Each machine has a varying hardware, a fixed cost, and an associated power and operational cost, the variable cost. To determine these costs, we received estimates from the OLCF. Additionally, we held operational cost constant across all four resources. Operational costs include mainly personnel. We subtracted the hardware costs of the GPUs, if any, since we did not use them. We also do not include file system costs in our calculations. The resulting fixed and variable costs for each architecture are summarized in Table II. We include a comparable cloud computing estimate using the Amazon Web Services (AWS) *i2.4xlarge* instance running Red Hat Linux. The comparable instance contains 16 virtualized CPU cores, 122 GB of RAM, 4×800 GB solid state drives and "High" I/O classification. Computationally, these types of nodes are likely to perform as their non-cloud counterparts [23]. We do not factor in data transfer costs or reserved-instance pricing when calculating the hourly rate for AWS. With this fixed and variable cost information and using equation (4), we can then plot the costs over time, as shown in Figure 2.

III. Benchmark Results

We report the run times of matrix multiplication, randomized SVD, linear regression and k-means in Figures 3, 4, 5 and 6, respectively. Additionally, we report the breakdown of the Initialization, I/O, Blockcyclic, Computation and Finalize times in Figure 7, to better understand the bottlenecks in an algorithm across the varying architectures. In general, we see that the Rhea (non-GPU) and Rhea (GPU) nodes significantly outperform across aspects like computation, I/O and network compared to traditional HPC clusters such as Titan and Eos.

A. Matrix Multiplication

The results of our matrix multiplication tests can be observed in Figure 3. Matrix multiplication, for two $n \times n$ matrices, has an algorithmic complexity of $O(n^3)$. This proves to be a computationally intensive algorithm. Because of this, we usually see 70% of the job time spent on computation (Figure 7).

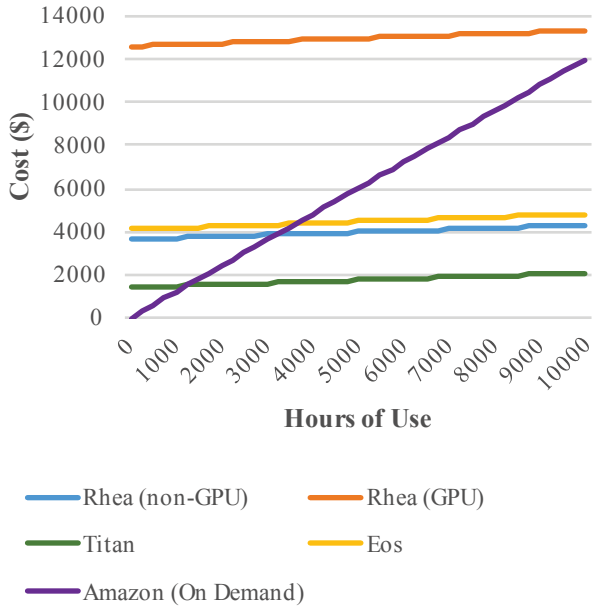

Fig. 2. Cost per hour of OLCF resources compared to AWS (per node)

B. Singular Value Decomposition

The results of running randomized SVD can be observed in Figure 4. We also break down the specific run times of Initialization, I/O, Blockcyclic, Computation and Finalize in Figure 7. We see that most of the time is split between computation and I/O. However, randomized SVD is a much faster algorithm than matrix multiplication: we spend much less time on computation. This leads to a higher percentage of time being spent on I/O.

C. Linear Regression

The results of running linear regression can be observed in Figure 5. Linear regression proved to be an extremely fast algorithm in comparison to the others tested. As such, more than half the time of every job, on every machine, was spent on the Finalize stage (Figure 7). Linear regression, through QR decomposition using Householder transformations, has an algorithmic complexity of $O(2nf^2 - \frac{2f^3}{3})$, where n is the number of observations and f is the number of features. Because our matrix factorization is taking place on an $n \times 2$ matrix, meaning $f = 1$, this keeps the number of computations low. As more and more predictors are added, particularly those which may be correlated, the slower QR factorization becomes.

D. K-Means

The results of running k-means can be observed in Figure 6. If we observe the differences across 1, 2 and 5 nodes, we also see that k-means scales well when more nodes are added. Since k-means is a computationally intensive and iterative algorithm, over 90% of its time was spent on computation alone.

TABLE II

NODE-LEVEL HOURLY COSTS OF OLCF RESOURCES

Architecture	Electrical Cost / hr	Operational Cost / hr	Total (Variable) Cost / hr	Hardware (Fixed) Cost
Rhea (non-GPU)	$0.0247	$0.0401	$0.0648	$3,650
Rhea (GPU)	$0.0315	$0.0401	$0.0717	$12,600
Titan	$0.0268	$0.0401	$0.0669	$1,415
Eos	$0.0326	$0.0401	$0.0727	$4,100
Amazon Web Services (On Demand)	N/A	N/A	$3.54	$0.00

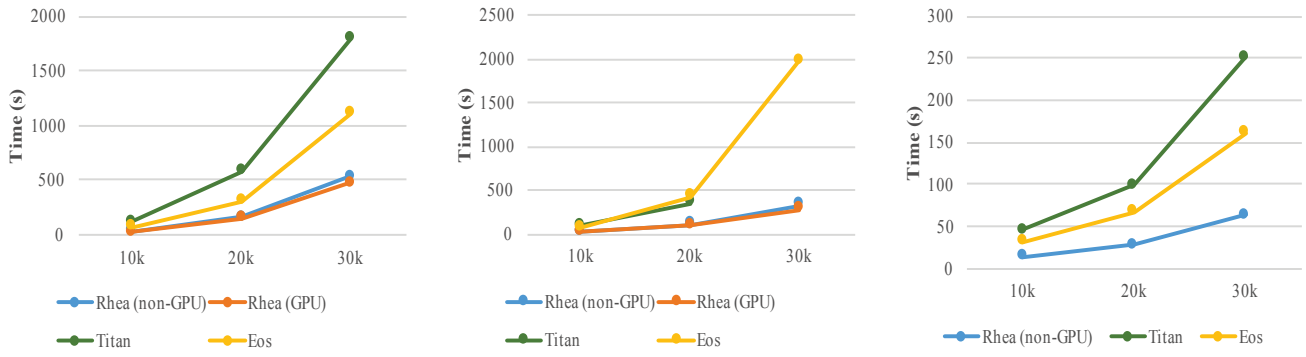

Fig. 3. Benchmark run times of 1, 2 and 5 node jobs for matrix multiplication by matrix size ($n \times n$)

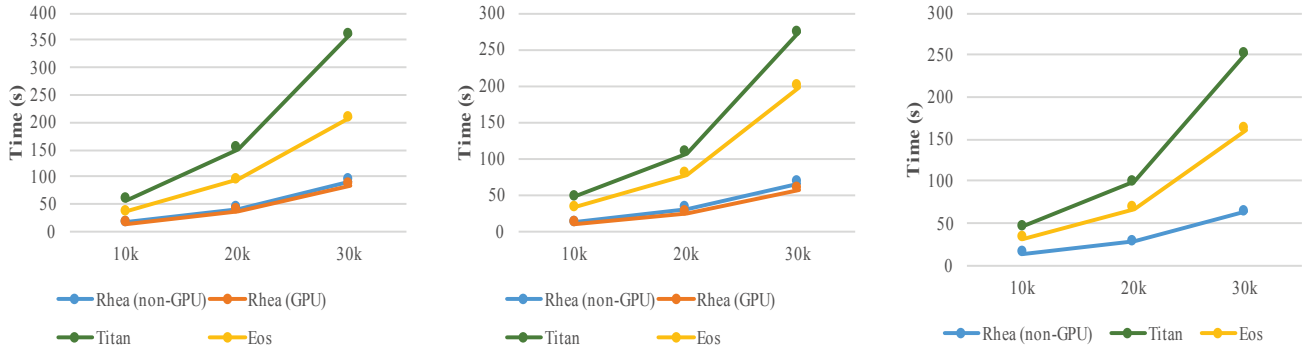

Fig. 4. Benchmark run times of 1, 2 and 5 node jobs for randomized SVD by matrix size ($n \times n$)

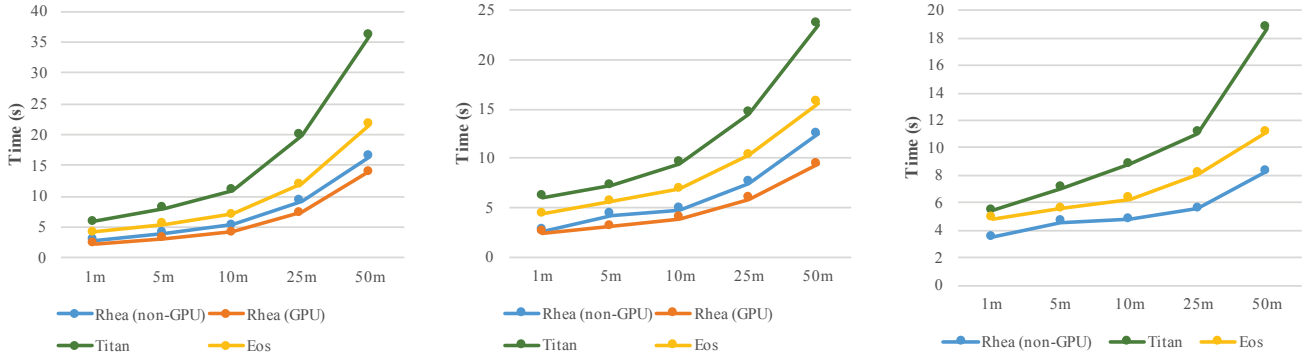

Fig. 5. Benchmark run times of 1, 2 and 5 node jobs for linear regression by number of observations

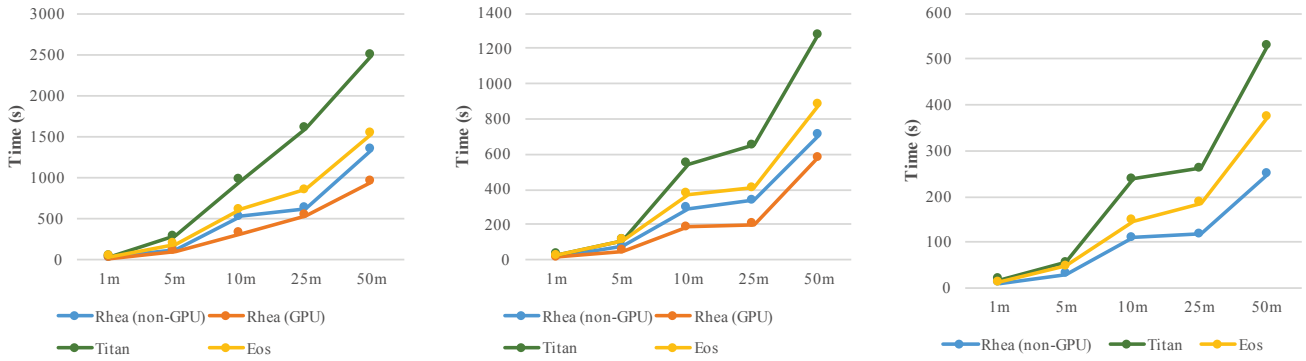

Fig. 6. Benchmark run times of 1, 2 and 5 node jobs for k-means by number of observations

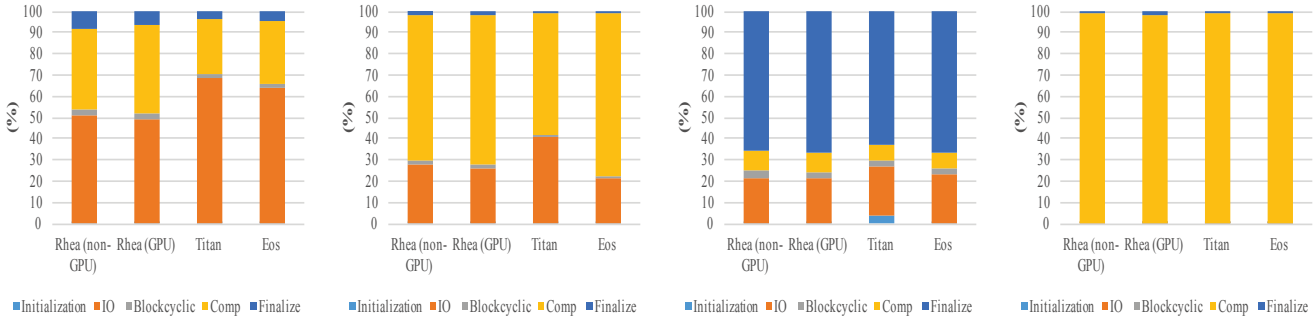

Fig. 7. Breakdown of run times for a $20,000 \times 20,000$ matrix for SVD and matrix multiplication and 10 million rows for linear regression and k-means

E. Spark vs. pbdR

One of the objectives of this study was to examine the performance differences between Spark and pbdR. We tested both linear regression and k-means on the same data sets used in our pbdR experiments. We did not test matrix multiplication or randomized SVD on Spark, because those algorithms are not included in MLlib. For each test, we used 5 compute nodes, in order to simulate a typical use case of Spark. Additionally, we used Spark on the Rhea (non-GPU) architecture, which further resembled typical Spark usage. We decided not to test on lower node counts, because Spark already uses a whole node as a "master" node, which does not compute, but rather controls the executor nodes. By testing 5 compute nodes, we are also able to make comparisons with our 5 node tests in pbdR. In MLlib, linear regression uses stochastic gradient descent to estimate β. This is different from pbdR's QR decomposition, which is a non-iterative method. K-means uses a parallelized variant of the k-means++ algorithm called k-means|| [24].

As we can tell from Figure 8, pbdR vastly outperforms Spark, especially at small data sizes. However, it is also observable that Spark may provide convenient scaling properties in both node count and data size. However, the study of these properties lies outside the scope of this paper. Additionally, Spark's RDD data structure provides fault tolerance in the event of hardware failure on nodes – something which pbdR does not provide.

TABLE III
PERFORMANCE SPEED LOSS BY NODE AND ALGORITHM

	Matrix Mult.	SVD	Linear Reg.	K-Means
Titan	3.80x	3.58x	2.51x	2.49x
Eos	2.22x	7.24x	1.68x	1.71x
Rhea (non-GPU)	1.09x	1.15x	1.28x	1.56x
Rhea (GPU)	1.00x	1.00x	1.00x	1.00x

IV. Discussion

From a performance perspective, it is clear that the Rhea (GPU) nodes perform the fastest, especially when compared to the traditional HPC hardware such as Titan or Eos. However, when compared to the cheaper Rhea (non-GPU) architecture, the speedup attained by the Rhea (GPU) machines is marginal. As indicated in Table III, we see that Rhea (GPU) achieved an average speedup of 3.1x vs. Titan and an average speedup of 3.2x vs. Eos. Against Rhea (non-GPU), Rhea (GPU) achieved only an average speedup of 1.3x.

A. Low Level Cache, Memory Bandwidth and I/O

Low level cache sizes, as well as memory bandwidth, are important considerations for the performance of data science applications. Because of the cache limitations among processors and the memory intensive nature of data science applications, these applications incur a higher floating point instructions (FLIPS) to L2 cache miss ratio (Fig. 9). There

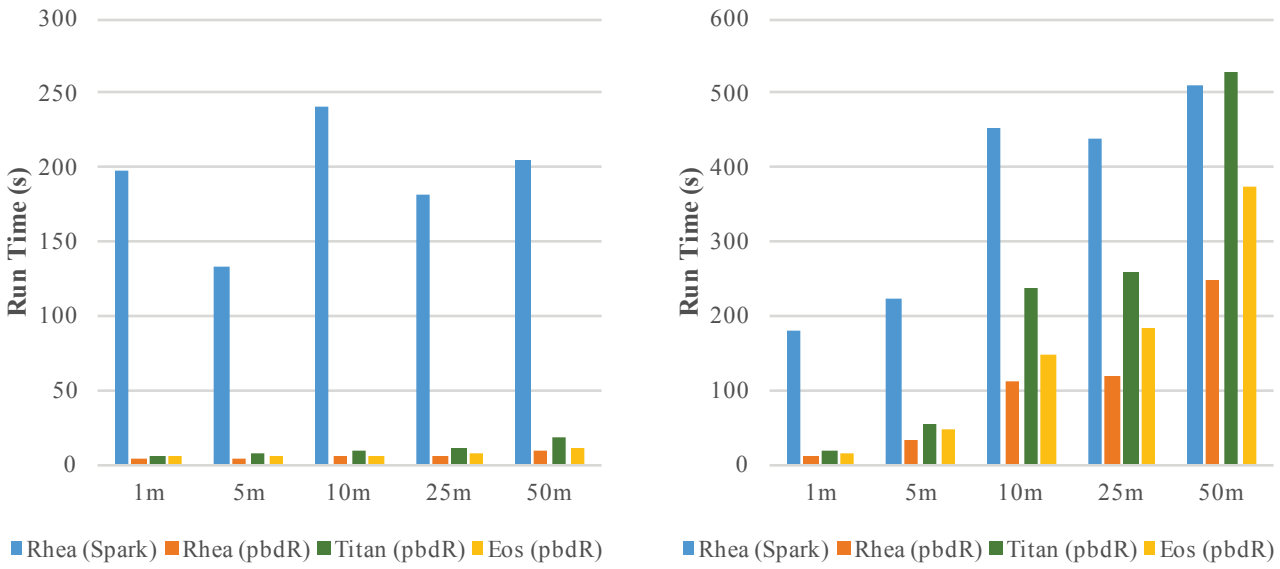

Fig. 8. Comparisons between pbdR and Spark on linear regression (left) and k-means (right)

TABLE IV
MONTE CARLO PI SIMULATION USING 10,000 POINTS

	Rhea (non-GPU)	Rhea (GPU)	Titan	Eos
Sim. Time (s)	0.929	0.735	0.432	0.471

is also a direct relationship between the size of the cache and the percentage of cache hits that are misses. Because of these cache misses, data must be pulled from memory, meaning data science applications may be memory bound. Although Titan exhibited slower compute times then both Rhea and Eos architectures, that is not enough evidence to determine that Titan's bottleneck was due to memory bandwidth. To determine if memory bandwidth was a bottleneck, we performed a Monte Carlo estimation of π by populating the first quadrant of a unit circle with 10,000 points. We then calculate the proportion of those that lie within the unit circle, which we can then evaluate π. This is an example of a program that has no significant memory requirements – data is generated and is small enough that it is kept in cache. In fact, it is an example of a compute bound program. This is highlighted in Figure 9, which shows the ratio of L2 Data Cache misses to floating point instructions (FLIPs). The results of this small experiment can be observed in Table IV, and show that in fact, Titan and Eos perform better than both of the Rhea architectures in this exercise. These results are expected, as both Titan and Eos were built with simulation tasks in mind.

Now, memory bandwidth becomes a key metric to look for in data science architectures. In Figure 10, we can see the memory bandwidth by thread count for OLCF resources. Consistent with the performance results in Figures 3-6, the Titan resource also exhibits the slowest memory bandwidth. The data for memory bandwidth was collected using the

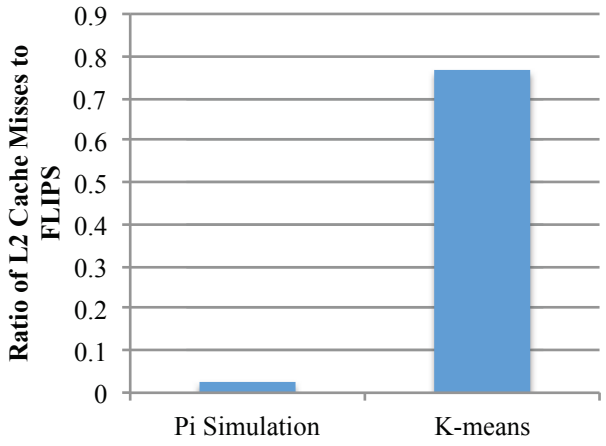

Fig. 9. Memory Bound Nature of Applications on Titan

STREAM benchmarking application [13].

Limitations of current HPC architectures include low RAM capacity, I/O bottlenecks and long queue times. Because of their low RAM capacity, HPC resources will require typically more nodes to keep data in memory, which will drive up cost-per-algorithm. Additionally, in Figure 7, we see that Titan and Eos, our representative HPC architectures, exhibit significant I/O bottlenecks, with about 60-70% of run time spent on I/O processes. These machines were not built with I/O read capabilities in mind – they were built primarily for simulation purposes. For simulation applications, I/O write capabilities are crucial, and the Lustre file system that these resources run upon is optimized for one way writes. This can pose problems for any hardware trying to perform big data analytics, as the data must be loaded into memory in the first place. Time spent on I/O was also significant

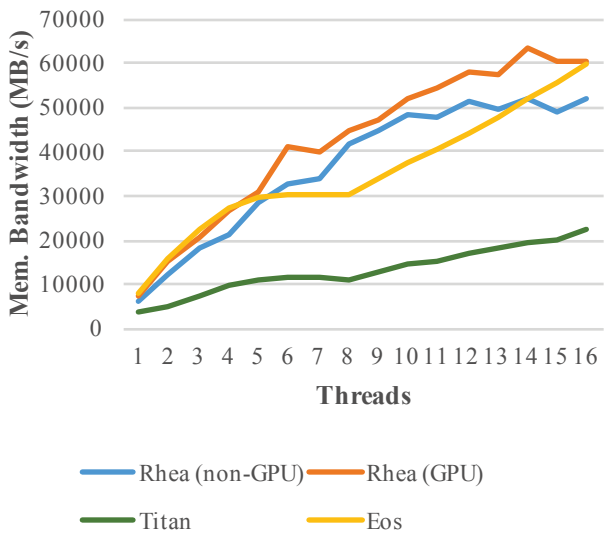

Fig. 10. OLCF memory bandwidth by thread count

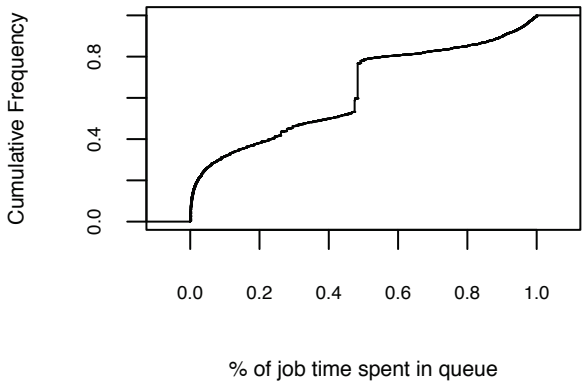

Fig. 11. Cumulative distribution function of the percent of job time in queue (Rhea)

for Rhea (non-GPU) and Rhea (GPU), which may indicate that focus on improving I/O read ability rather than marginal increases in computation speed may yield great speedups in total run time for data science algorithms.

B. Queuing Systems

Queuing systems and policies are often a hidden driver behind the total job time on computing resources, and therefore, a hidden contributor to "total" job performance. Many times, institutions impose queuing policies based on nodes requested, wall time, and user core-hour budgets.

Traditionally, HPC machines institute a batch queuing system much different than the on demand nature of data science. For data science workloads, particularly small ones, such policies and systems detract from providing on-demand analytics. As evidenced in Figure 11, about half of the jobs on Rhea spend half of their total job time waiting in the queue. The average run time for these jobs is 95 minutes. For Rhea, where the average wait time for jobs under 100 nodes is 52 minutes, we lose the ability to perform analytics on-demand. Because of this, if the data is small enough and interactivity is a requirement, small data science workloads may better take place on a local machine or an AWS instance.

A direct remedy for the on-demand problem exists in providing more resources to the users through more fixed cost by scaling out existing computing resources. A less investment intensive solution may include decentralization of computing resources among different constituent users in a "to each their own" fashion.

C. Cost Analysis

In order to gain performance, whether through increased memory throughput at the node level, or by scaling out resources to decrease queue times at the system level, there is an associated cost. Memory, especially for "fat" nodes, is not inexpensive. For the Rhea (GPU) nodes, memory attributes over 40% of total fixed cost. Queue systems can also contribute to significant resource waste, and thus cost, for HPC systems. Typically, HPC queue systems will free up cores for large but short lived simulation jobs. The process of freeing up cores can contribute to many wasted core hours, which essentially drives up cost per data science workload, since fewer data science workloads can be completed.

The question then becomes: what is a data science workload? Let us start by considering a "unit" of analytics as a collection of analytics jobs and define this unit of analytics as:

- 1,000 matrix multiplications on $20,000 \times 20,000$ matrices
- 1,000 singular value decompositions on $30,000 \times 30,000$ matrices
- 1,000 linear regressions on 50 million observations
- 1,000 k-means clustering on 50 million observations with 2 features
- Using 2 nodes

We can then use our benchmarking results and cost information to determine how much this "unit" of analytics costs and how long it will take. We plot this information in Figure 12, with the x-axis as units of analytics, the y-axis as cost of that many units, and the size of the bubble being the run time of units. We see that, at a small scale, Titan provides more cost savings than the other resources, while slightly lacking in performance. As more and more units of analytics are added, we see that the Rhea (non-GPU) resource in fact overtakes all others in terms of cost. Additionally, the performance of Rhea (non-GPU) at scale is also better than that of its closest "competitors", Titan and Eos.

One must also keep in mind the size of the data when considering cost of data analytics resources. It is imperative that the data being analyzed can fully fit into memory; therefore, memory capacity is crucial for data science workload

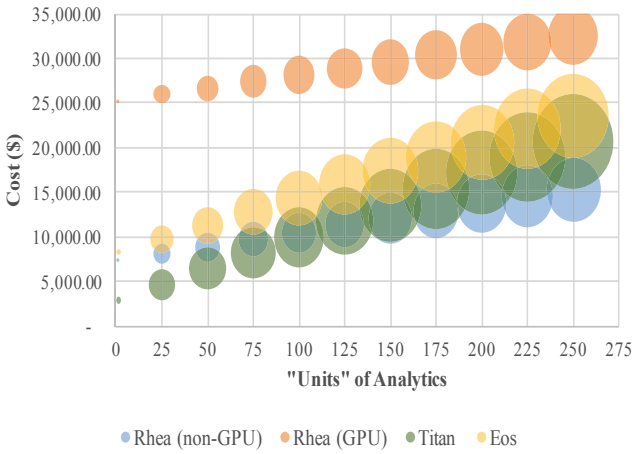

Fig. 12. Relationship between quantity of analytics, cost and performance

TABLE V
COST OF DATA WORKLOAD BY SIZE

	Data			
	100 GB	1 TB	10 TB	100 TB
Rhea				
Nodes	3	24	235	2,344
Fixed Cost ($)	10,950	87,600	857,750	8,555,600
Hourly Cost	0.194	1.555	15.228	151.891
Rhea (GPU)				
Nodes	1	3	30	293
Fixed Cost	12,600	37,800	378,000	3,691,800
Hourly Cost	0.072	0.215	2.151	21.008
Titan				
Nodes	10	94	938	9375
Fixed Cost	14,150	133,010	1,327,270	13,265,625
Hourly Cost	0.669	6.289	62.752	627.188
Eos				
Nodes	5	47	469	4688
Fixed Cost	20,500	192,700	1,922,900	19,220,800
Hourly Cost	0.364	3.417	34.096	340.818
i2.4xlarge Instance				
Nodes	1	24	235	2,344
Fixed Cost	0	0	0	0
Hourly Cost	3.382	79.379	753.679	7,434.313

efficiency. If one is dealing with a 100 GB data set, it does not mean that one only needs 100 GB of RAM. We believe requesting enough nodes to have about three times the data size is a suitable amount, as we need extra memory to shuffle around and create new instances of the data, as well as to maintain the operating system of each node. We can see in Table V that the both the fixed and hourly cost far outpaces those of large node memory architectures, such as the Rhea architectures. Additionally, as observed earlier, fast memory bandwidth is essential for data science applications. This is especially true when the data is being broken into larger chunks to be spread across fewer nodes.

D. Limitations

There are several limitations of this study. One of the most glaring is related to the varying costs associated with computing. Because these tests were carried out at ORNL, where electricity is relatively inexpensive, the results may not carry over to areas with higher electricity costs. Additionally, we did not focus on testing the GPU or interconnect in this study. As the GPU constitutes substantial cost of certain machines, it increases cost with no return if not utilized. Unfortunately, due to the expertise needed to leverage the power of GPUs effectively, this may be a common problem among institutions. A strong interconnect may bring serious speedups to iterative algorithms which frequently communicate data cross nodes. Further research is needed to see how the GPU and interconnect fit into the data science computing puzzle. Finally, we do not test sparse matrix algorithms, which may show quite different benchmark results and scale differently that those that use dense data sets.

V. CONCLUSION

In this paper, we explored the performance and cost characteristics of HPC infrastructures for data science. On the performance side, we have found that "fat" node structures, with large amounts of memory and high memory bandwidth, are better suited for big data analytics, delivering up to 3x speedup. From our cost analysis, we also conclude that due to their flexibility and availability, cloud computing infrastructures are best suited to small or experimental jobs. Cost at scale for data analytics favors HPC. We also conclude that on traditional HPC infrastructures, queuing mechanisms are a significant roadblock to the time-sensitive nature of data science. More research is needed to understand how queue systems may work better for data science job scheduling. Finally, we see that pbdR gives significant speedup on data science algorithms when compared to Apache Spark on HPC architectures. Future research will be directed towards building a comprehensive cost-performance model, as well as into understanding and quantifying the role of other hardware components, such as GPU and interconnect, in data science applications.

ACKNOWLEDGMENT

We would like to thank George Ostrouchov for his help on understanding pbdR (http://r-pbd.org). We would also like to thank Vladimir Protopopescu for his help in editing this paper.

REFERENCES

[1] OLCF, "Titan cray xk7," Aug 2016.
[2] D. Talia, "Clouds for scalable big data analytics," *Computer*, vol. 46, pp. 98–101, May 2013.
[3] G. Ostrouchov and N. F. Samatova, "High end computing for full-context analysis and visualization: when the experiment is done," in *High End Computing Revitalization Task Force*, Jun 2003.
[4] A. Guazzelli, K. Stathatos, and M. Zeller, "Efficient deployment of predictive analytics through open standards and cloud computing," *SIGKDD Explor. Newsl.*, vol. 11, pp. 32–38, Nov 2009.
[5] P. Wong, H. Shen, C. Johnson, C. Chen, and R. Ross, "The top 10 challenges in extreme-scale visual analytics," *IEEE Computer Graphics and Applications*, vol. 32, no. 4, pp. 63–67, 2012.
[6] L. A. Barroso, "The price of performance," *Queue*, vol. 3, pp. 48–53, Sep 2005.

[7] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–12, May 2009.

[8] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," in *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, Nov 2008.

[9] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: A viable solution?," in *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*, DADC '08, (New York, NY, USA), pp. 55–64, ACM, 2008.

[10] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *Login*, vol. 33, pp. 18–23, oct 2008.

[11] A. G. Carlyle, S. L. Harrell, and P. M. Smith, "Cost-effective hpc: The community or the cloud?," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 169–176, Nov 2010.

[12] OLCF, "System user guides," Aug 2016.

[13] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[14] 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*, NSDI'12, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2012.

[15] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, pp. 1235–1241, Jan 2016.

[16] G. Ostrouchov, W.-C. Chen, D. Schmidt, and P. Patel in *Programming with Big Data in R*, (Stanford, CA, USA), 6th Extremely Large Databases Conference (XLDB), Aug 2012.

[17] W.-C. Chen and G. Ostrouchov, "A quick guide for the pmclust package," 2016.

[18] D. Schmidt, W.-C. Chen, G. Ostrouchov, and P. Patel, "Guide to the pbddmat package," 2015.

[19] M. T. Heath and C. H. Romine, "Parallel solution of triangular systems on distributed-memory multiprocessors," *SIAM J. Sci. Stat. Comput.*, vol. 9, pp. 558–588, May 1988.

[20] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[21] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, (New York, NY, USA), pp. 36–47, ACM, 2011.

[22] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, "Big data analytics: a survey," *Journal of Big Data*, vol. 2, no. 1, pp. 1–32, 2015.

[23] I. Sadooghi, J. H. Martin, T. Li, K. Brandstatter, K. Maheshwari, T. P. P. de Lacerda Ruivo, G. Garzoglio, S. Timm, Y. Zhao, and I. Raicu, "Understanding the performance and potential of cloud computing for scientific applications," in *The Sixth International Workshop on Data-Intensive Computing in the Clouds*, 2015.

[24] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proc. VLDB Endow.*, vol. 5, pp. 622–633, Mar. 2012.