Related work to this research are the efforts by industry and academia on defining benchmarks and Big Data standards. We discuss benchmarks closely in chapter 0 – Landscape of Benchmarks and reserve this space to recognize the work of Big Data benchmarking community (BDBC) that is working towards the development of industry standard benchmarks for Big Data. BDBC and its workshop series on Big Data benchmarking (WBDB) are the closest efforts in the literature on the subject we could find besides Ogre efforts.

**Database Benchmarking**

Following the experience gathered from the first WBDB workshop, Baru et al. [1] discusses the need for coming up with systematic approaches to Big Data benchmarking and presents 5 important design considerations – component vs. end-to-end, Big Data applications, data sources, scaling, and metrics – as described below.

* **Component vs. end-to-end**

Component benchmarks, for example SPEC’s CPU benchmark [2] and TeraSort [3], evaluate only one or few entities of the system. These are relatively easy to define, especially when components have standard application programming interfaces (APIs). End-to-end benchmarks test the entire system rather picking up a particular component, which usually involves tackling a complex setup and evaluation. TPC-DS [4] and BigBench [5] are examples for such benchmarks where they measure the system’s ability from data loading to data serving for different queries over a variety of data types while supporting periodic updates.

* **Big Data applications**

The first thing to note with Big Data applications is their diversity ranging from scientific to social data, thus making it impossible to find a single application to Big Data. The other question is deciding if a particular benchmark should model a concrete application or an abstract model. The authors claim that modeling a concrete application has the advantages of being able to provide a detailed specification and to relate real world business workloads to the benchmark. They also note that such modeling could be done only by considering the key characteristics of the particular application.

* **Data sources**

The third consideration is whether to use real data or synthetic data for benchmarks and the conclusion has been to use synthetic data that ensures the characteristics of real data. The reasons are, 1) synthetic data sets are relatively easy to generate on different scales 2) real data sets may not reflect all the properties of data expected to be handled by the application, and 3) synthetic data could incorporate characteristics of real data.

* **Scaling**

The system under test (SUT) and the real installation may be different in scale and with Big Data it is most likely that the latter is much larger than SUT in contrast to the case with traditional TPC tests where usually the opposite is true. One aspect of scaling is being able to extrapolate the benchmark results from SUT to reason about the real system. Another aspect is dealing with “elasticity”, especially in cloud environments where data sizes and resources could grow and shrink dynamically. It should also take into consideration the failures and provide benchmarking measures.

* **Metrics**

Benchmark metrics usually target for performance, but price for performance is equally important, especially with Big Data hardware and software systems. The suggestion is to adopt the TPC’s pricing scheme and also include energy costs and system setup costs. The latter could be very large for Big Data systems that they may not be ignored.

In a recent tutorial [6] Baru et al. extend these to include ideas such as reuse, where to get data, verifiability, and implementation based vs paper and pencil. They also introduce the “Vs” of Big Data benchmarks – volume, velocity, and variety. Volume determines if the benchmark can test the scalability of the system to large volumes of data. Velocity questions the ability to deal with frequent data updates and variety is the ability to include operations on heterogeneous data.

Another point to take from [6] is the types of benchmarks – micro, functional, genre-specific, and application level – as described below.

* **Micro-benchmarks**

These are benchmarks designed to evaluate specific low level system operations. Examples of micro-benchmarks include the popular OSU [7] benchmarks suite and enhanced DFSIO from HiBench [8].

* **Functional benchmarks**

These may also be called as component benchmarks where the objective is to exercise a particular high level function such as sorting or an individual SQL operation like select, project, join, etc.

* **Genre-specific benchmarks**

Any benchmark that is related with the type of data such as in the case of Graph500 [9] where it deals with graph data.

* **Application level benchmarks**

These are full applications that evaluates hardware and software for a given data and workload.

**Existing Benchmarks**

The intent of this section is to present an overview of the existing benchmarks, especially those related with Big Data applications, as a guide and to emphasize the need of yet another benchmark to cover the problems in question. We will start the discussion with HPC benchmarks.

**HPC Benchmarks**

***LINPACK and Its Variants***

The bloodline of LINPACK [10] includes its shared memory version – LAPACK [11], the parallel distributed memory implementation – ScaLAPACK [12], and the Top500’s [13] yardstick – HPL [14, 15]. These are kernel solvers for dense linear systems of the form $Ax=b$. The strategy is to use lower upper (LU) factorization followed by a solver that totals $2n^{3}/3+2n^{2}$ floating point operations (*flop*). The performance metric is *flop* per second – generally mega or giga *flop*s per second (Mflop/s of Gflop/s).

The LINPACK benchmark report [16] includes results from three benchmarks – LINPACK Fortran n=100, LINPACK n=1000, and HPL. The first is a sequential Fortran based solver for a matrix of order 100. The rules specify that no change other than compiler optimizations are allowed for this case. The second benchmark is for a matrix of order 1000 with relaxed rules where the user can replace the LU factorization and solver steps. The report also includes results exploiting shared memory parallelism in a fork-join style for the n=1000 test. HPL benchmark relaxes both the choice of implementation and problem size. Its parallel algorithm for distributed memory systems is explained in [17] and a scalable implementation is packaged into the HPL software distribution that scales both with respect to the amount of computation and communication volume as long as the memory usage per processor is maintained [18].

***NAS Parallel Benchmarks***

NAS Parallel Benchmarks (NPB) are a set of kernel and pseudo applications derived from computational fluid dynamics (CFD) applications. They are meant to compare the performance of parallel computers and to serve as a standard indicator of performance [19]. The original NPB 1, which is a paper and pencil specification, includes 5 kernels and 3 pseudo applications. Optimized message passing interface (MPI) parallel implementations became available since version 2.3.

The original benchmark set was extended with a multi zone (MZ) implementations of the original block tridiagonal (BT), scalar pentadiagonal (SP), and lower upper (LU) pseudo applications. MZ versions intend to exploit multiple levels of parallelism and the implementations use MPI plus threading with OpenMP [20]. NPB was further extended to include benchmarks that evaluate unstructured computation, parallel I/O, and data movement. Parallel to NPB another set of benchmarks were introduced as GridNPB to rate the performance of grid environments.

A notable feature in NPB is its well defined benchmark classes – small (S), workstation (W), standard, and large. Standard class is further divided into sub classes A, B, and C with problem size increasing roughly 4 times from going one class to the next. The large class also introduce D, E, and F sub classes where the problem size increase is roughly 16 times. A detailed description of the actual problem sizes for each class is available in [21] and we would like to capture this property in our proposed benchmarking strategy as well.

**Big Data Benchmarks**

There is a range of Big Data benchmarks in the current literature and we intend to describe a selected few covering different areas in this section.

***BigDataBench***

BigDataBench [22, 23] is a benchmark suite targeting Internet services. There is a total of 33 benchmarks (or workloads as the authors refer) implemented out of 42 and they are classified into 5 application domains – search engine, social network, e-commerce, multimedia data analytics, and bioinformatics. Moreover, these 33 benchmarks have multiple implementations for some of them, thus totaling 77 tests. The implementations use several components of the Apache Big Data Stack (ABDS) [24, 25] and some of their commercial adaptations. An extracted summary of benchmarks from [23] is given in Table 3.

The latest (version 3.1) handbook [23] of the BigDataBench mentions that each workload is quantified over 45 micro-architectural level metrics from the categories instruction mix, cache behavior, TLB behavior, branch execution, pipeline behavior, offcore request, snoop response, parallelism, and operation intensity. The original paper [22] also presents 3 user perceivable metrics – processes requests per second (RPS), operations per second (OPS), and data processes per second (DPS). Note, each of these are relevant only for some workloads.

Table 3 Benchmark summary of BigDataBench

|  |  |  |
| --- | --- | --- |
| **Application Domain** | **Operation or Algorithm** | **Software Stack** |
| Search Engine | Sort, Grep, WordCount, Index, PageRank, Nutch Server, Read, Write, Scan | Hadoop, Spark, MPI, Nutch, HBase, MySQL |
| Social Network | K-means, Connected Components (CC), BFS | Hadoop, Spark, MPI |
| E-commerce | Select, Aggregate, and Join queries, Collaborative Filtering (CF), Naïve Bayes, Project, Filter, Cross Product, OrderBy, Union, Difference, Aggregation | Impala, Hive, Shark |
| Multimedia | BasicMPEG, SIFT, DBN, Speech Recognition, Image Segmentation, Face Detection |  MPI |
| Bioinformatics | SAND, BLAST | Work Queue, MPI |

BigDataBench presents two things – implications of data volume and benchmark characterization. The paper [22] presents the importance of testing with increasing loads to figure out the performance trends in each case. The metrics, million instructions per second (MIPS) and cache misses per 1000 instructions (MPKI) are given to elaborate this fact. The benchmark characterization measures operation intensity and effects of hierarchical memory. In conclusion they present that the kind of benchmarks tested in BigDataBench show relatively low ratios of computation to memory accesses compared to traditional HPC benchmarks. Further, they show that L3 caches show the least MPKI numbers for these benchmarks and that a possible cause of seeing higher MPKI values in lower level caches (L1, L2) could be due to the use of deep software stacks.

BigDataBench besides providing a large number of benchmarks and metrics, also presents a way to reduce the number of benchmarks that one would need to run in order to assess a system comprehensively. The strategy behind this is instead of characterizing a benchmark into 45 (micro-architectural metrics) dimensions, pick the most uncorrelated dimensions with the help of running principal component analysis (PCA) [26] and then cluster the benchmark performance vectors with K-means clustering to form groups of similar benchmarks. Then pick a representative benchmark from each cluster either by picking on close to the edge of a cluster or the mid of a cluster. There are two lists of such shortlisted benchmarks presented in [23].

***HiBench***

HiBench [8] is a Hadoop benchmark suite intended to evaluate MapReduce styled applications. It identifies the interest in the community to use Hadoop and its ecosystem – Pig, Hive, Mahout, etc. – to areas such as machine learning, bioinformatics, and financial analysis. The introduction of HiBench, as it authors claim, is to overcome the limited representation and diversity of existing benchmarks for Hadoop at its time. The benchmarks they have compared are sort programs, GridMix [27], DFSIO [28], and Hive performance benchmark [29]. A few reasons why these does not do a fair evaluation are, 1) does not exhibit computations compared to real applications, 2) no data access outside map tasks, and 3) represents only analytical database queries (Hive benchmarks), which does not evaluate MapReduce over a broad spectrum of large data analysis.

HiBench introduces micro-benchmarks and real world applications. The micro-benchmarks include the original Sort, WordCount, and TeraSort from Hadoop distribution itself. The real applications are Nutch indexing, PageRank, Bayesian classification, K-means clustering, and EnhancedDFSIO. The latter could be identified as a micro-benchmark in today’s context and is an extension on the original DFSIO to include measure aggregated I/O bandwidth. HiBench evaluates these benchmarks for job running time and throughput, aggregated HDFS bandwidth, utilization of CPU, memory and I/O, and data access patterns, i.e. data sizes in map-in, map-out/combiner-in, combiner-out/shuffle-in, and reduce out stages. In conclusion the authors claim HiBench represents a wider range of data analytic problems with diverse data access and resource utilization patterns. Latest release for HiBench is version 3.0 done on October 2014 and is available at [30].

***Graph500***

Graph500 [31], unlike other Big Data benchmarks, is intended to evaluate a variety of architectures, programming models, and languages and frameworks against data intensive workloads. It brings to light the point that systems targeted for traditional physics simulations may not be the best for data intensive problems. The benchmark performs breadth-first graph search and defines 6 problem classes denoted as levels 10 through 15. These indicate the storage in bytes required to store the edge list such that for a given level, $L$, the size will be in the order of $10^{L}$.

There are 2 timed kernels in Graph500 – kernel 1 creates a graph representation from an edge list and kernel 2 performs the BFS. Kernel 2 is run multiple times (64 times usually) each with a different starting vertex. After each run a soft validation is run on results. The soft validation checks for properties of a correct BFS tree rather verifying if the resultant BFS tree is the one for the input graph and the particular starting vertex. The performance metric of Graph500 defines a new rate called traversed edges per second, $TEPS=m/time\_{k2}$, where $m$ is the number of edges including any multiple edges and self-loops, and $time\_{k2}$ is the kernel 2’s execution time.

***BigBench***

BigBench [5, 32] is an industry lead effort to defining an comprehensive Big Data benchmark that emerged with a proposal that appeared in the first workshop on Big Data benchmarking (WBDB) [33]. It is a paper and pencil specification, but comes with a reference implementation to get started. BigBench models a retailer and benchmarks 30 queries around it covering 5 business categories depicted in the McKinsey report [34].

The retailer data model in BigBench address the three V’s – volume, variety, and velocity – of Big Data systems. It covers variety by introducing structured, semi-structured, and unstructured data in the model. While the first is an adaptation from the TPC-DS [4] benchmark’s data model, the semi-structured data represents the click stream on the site, and unstructured data denotes product reviews submitted by users. Volume and velocity are covered with a scale factor in the specification that determines the size for all data types, and a periodic refresh process based on TPC-DS’s data maintenance respectively.

Part of the BigBench research is on data generation, which includes an extension to the popular parallel data generation framework (PDGF) [35] to generate the click stream data (semi-structured), and a novel synthetic reviews (unstructured text data) generator, TextGen, which is seamlessly integrated with PDGF.

There are a total of 30 queries covering 10 classes from 5 business categories. While these cover the business side well, they also cover 3 technical dimensions – data source, processing type, and analytic technique. Data source coverage is to represent all three – structured, semi-structured, and unstructured data – in the queries. Given that BigBench is a paper and pencil specification, the queries are specified using plain English. While some of these could be implemented efficiently with structured query language (SQL) or Hive-QL [36] like declarative syntaxes, the others could benefit from a procedural based implementation like MapReduce or a mix of these two approaches. The processing type dimension assures that the queries make a reasonable coverage of these three types. BigBench identifies 3 analytic techniques in answering queries – statistical analysis, data mining, and simple reporting. The analytic technique dimension of BigBench does justice to these 3 techniques by covering them reasonably in the 30 queries. The paper leaves out defining a performance metric for future work, but suggests taking a geometric mean approach as $\sqrt[30]{\prod\_{i=1}^{30}P\_{i}}$ where $P\_{i}$ denotes execution time for query $i$. It also presents their experience implementing and running this end-to-end on Teradata Aster database management system (DBMS).

In summary, BigBench is in active development at present and provides a good coverage on business related queries over a synthetic dataset. Additionally, plans are set for a TPC proposal with its version 2.0 besides being a benchmark on its own.

***LinkBench***

LinkBench [37] is a benchmark developed at Facebook to evaluate its graph serving capabilities. Note, this evaluates a transactional workload, which is different from a graph processing benchmark like Graph500 that runs an analytic workload. LinkBench is intended to serve as a synthetic benchmark to predict the performance of a database system serving Facebook’s production data, thereby reducing the need to perform costly and time consuming evaluations mirroring real data and requests.

The data model of LinkBench is a social graph where nodes and edges are represented using appropriate structures in the underlying datastore, for example using tables with MySQL. The authors have studied in detail the characteristics of the Facebook’s data when coming up with a data generator that would closely resemble it.

The workload is also modeled after careful studying of actual social transactions. They consider several factors such as access patterns and distributions, access patterns by data type, graph structure and access patterns, and update characterization in coming up with an operation mix for the benchmark.

The design includes a driver program that generates data and fires up requester threads with the operation mix. The connections to the data store are handled through LinkBench’s graph store implementation, which currently includes support for MySQL back ends. Most of the information for the benchmark is fed through simple configuration file, which makes it easy to adapt for different settings in future.

Primary metrics included in the benchmark are operation latency and mean operation throughput. The other metrics include price/performance, CPU usage, I/O count per second, I/O rate MB/s, resident memory size, and persistent storage size.

***MineBench***

MineBench [38] is a benchmark targeted for data mining workloads and presents 15 applications covering 5 categories as shown in Table 4.

Table 4 MineBench applications

|  |  |  |
| --- | --- | --- |
| **Application** | **Category** | **Description** |
| ScalParC | Classification | Decision tree classification |
| Naïve Bayesian | Classification | Simple statistical classifier |
| SNP | Classification | Hill-climbing search method for DNA dependency extraction |
| Research | Classification | RNA sequence search using stochastic Context-Free Grammars |
| SVM-RFE | Classification | Gene expression classifier using recursive feature elimination |
| K-means | Clustering | Mean-based data partitioning method |
| Fuzzy K-means | Clustering | Fuzzy logic-based data partitioning method |
| HOP | Clustering | Density-based grouping method |
| BIRCH | Clustering | Hierarchical Clustering method |
| Eclat | Association Rule Mining | Vertical database, Lattice transversal techniques used |
| Apriori | Association Rule Mining | Horizontal database, level-wise mining based on Apriori property |
| Utility | Association Rule Mining | Utility-based association rule mining |
| GeneNet | Structure Learning | Learning Gene relationship extraction using microarray-based method |
| SEMPHY | Structure Learning | Learning Gene sequencing using phylogenetic tree-based method |
| PLSA | Optimization | DNA sequence alignment using Smith-Waterman optimization method |

It has been a while since MineBench’s latest release in 2010, but it serves as a good reference for the kind of applications used in data mining. Moreover, these are real world applications and the authors provide OpenMP based parallel versions for most of them. The input data used in these applications come from real and synthetic data sets of varying size classes – small, medium, and large.

A performance characterization of data mining applications using MineBench is given in separate papers [39, 40]. The architectural characterization paper [40], in particular is interesting for a couple of reasons. First, it justifies the need to introduce a new benchmark system by identifying the diversity of data mining applications. It does so by representing each application as a vector of its performance counters and using K-means clustering to group them. While applications from other benchmarks such as SPEC INT, SPEC FP, MediaBench and TPC-H tend to cluster together, data mining applications falls under multiple clusters. Second, it characterizes the applications based on 1) execution time and scalability, 2) memory hierarchy behavior, and 3) instruction efficiency. While it is expected from any benchmark to have a study of performance and scalability, we find the other two dimensions are equally important and adoptable towards studying Big Data benchmarks as well.

***BG Benchmark***

BG [41] emulates read and write actions performed on a social networking datastore and benchmarks them against a given service level agreement (SLA). These actions originate from interactive social actions like view profile, list friends, view friend requests, etc. BG defines a data model and lists the social actions it benchmarks in detail in [41]. It introduces two metrics to characterize a given datastore as given below.

* **Social Action Rating (SoAR)**

Defines the highest number of completed actions per second agreeing to a given SLA.

* **Socialites**

Defines the highest number of simultaneous threads that issue requests against the datastore and satisfy the given SLA.

An SLA requires that for some fixed duration 1) a fixed percentage of requests observing latencies equal or less than a given threshold, and 2) the amount of unpredictable data is less than a given threshold. Quantifying unpredictable data is an offline process done through log analysis at the granularity of a social action.

BG implementation consists of three components – BG coordinator, BG client, and BG visualization deck. There can be multiple clients and they are responsible for data and action generation. The coordinator communicates with clients to instruct on how to generate data and emulate actions based on the given SLA. It also aggregates the results from clients and presents to the visualization deck for presentation.

**MDS and Clustering Ogres**

We briefly introduce about MDS and clustering Ogres, and their implementations in the SPIDAL package. We follow up on this with an introduction to real use cases of MDS and clustering in our work and preliminary performance results we have obtained.

**MDS**

MDS is a technique used often to visualize higher dimensional data in 3D. The input to an MDS algorithm is an $NxN$ distance matrix corresponding to the pairwise distances of $N$ item. The goal of the algorithm is to map each item as a point in the given lower dimension such that for any given pair of items the distance between them in the mapped space preserves the corresponding distance in the original space as shown in Figure 4.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|

|  |  |  |  |
| --- | --- | --- | --- |
| $$δ\_{11}$$ | $$δ\_{12}$$ | … | $$δ\_{15}$$ |
| $$δ\_{21}$$ | $$δ\_{22}$$ | … | $$δ\_{25}$$ |
| … | … | … | … |
| $$δ\_{51}$$ | $$δ\_{52}$$ | … | $$δ\_{55}$$ |

 |  |  |
|  |  |  |

Figure 4 MDS mapping of distances to points

Table 1 MDS instances in SPIDAL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Optimization Method** | **Language** | **Parallel Implementations** | **Target Environments** |
| MDSasChisq | Levenberg–Marquardt algorithm | C# | Message passing with MPI.NET [42] and threads | Windows HPC cluster |
| Java | Message passing with OpenMPI (Java binding) and threads | Linux cluster |
| Twister DA-SMACOF | Deterministic annealing | Java | Twister [43] iterative MapReduce | Cloud / Linux cluster |
| Harp DA-SMACOF | Deterministic annealing | Java | Harp Map-Collective framework [44] | Cloud / Linux cluster |
| Spark DA-SMACOF (ongoing) | Deterministic Annealing | Java / Scala | Spark [45] | Cloud / Linux cluster |
| MPI DA-SMACOF (ongoing) | Deterministic Annealing | Java | Message passing with OpenMPI (Java binding) [46] and threads | Linux cluster |

|  |  |  |
| --- | --- | --- |
| $$σ\left(X\right)=\sum\_{i<j\leq N}^{}w\_{ij}(d\_{ij}(X)-δ\_{ij})^{2}$$ | Equation 1

|  |
| --- |
|  |

 |

SPIDAL has a few instances of MDS Ogres given in Table 1 that minimizes Equation 1, where indices $ij$ indicate the pair of items, $w\_{ij}$ is the corresponding weight term, $δ\_{ij}$ is the distance in original space, and $d\_{ij}(X)$ is the distance in mapped space. Features of these instances include arbitrary weights, missing distances, and fixed points. Algorithmically, the entries in Figure 4 MDS mapping of distances to pointsFigure 4 MDS mapping of distances to points all into two categories – MDSasChisq uses Levenberg-Marquardt [47] to solve non-linear least squares problem in Equation 1, while others use SMACOF with deterministic annealing (DA) [48]. The latter technique finds the solution efficiently than MDSasChisq and hence the multiple parallel implementations.

**Clustering**

Table 2 Clustering instances in SPIDAL

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Optimization Method** | **Space** | **Language** | **Parallel Implementations** | **Target Environments** |
| DA-PWC | Deterministic annealing | Metric | Java | Message passing with OpenMPI (Java binding) [46] | Linux cluster |
| DA-VS | Deterministic annealing | Vector | Java | Message passing with OpenMPI (Java binding) [46] | Linux cluster |

We ship two clustering Ogre instances (see Table 2) that use the same DA optimization technique, but applied in clustering [49] – DA pairwise clustering (DA-PWC) [50] and DA vector sponge (DA-VS) [51, 52]. The first operates on metric space taking an $NxN$ pairwise distance matrix and mapping each of the $N$ points to a cluster. This has been the workhorse in our data analyses so far (see section 4.3) and we have a hybrid implementation based on OpenMPI Java bindings and threads. We also carry a legacy C# implementation on MPI.NET and threads targeted for Windows HPC clusters.

DA-VS is a recent introduction that performs clustering of vector data, again based on the DA optimization. It is still in early production stages and we offer OpenMPI Java plus thread based implementation and a legacy C# based version similar to DAPWC.

**MDS and Clustering Use Cases**

We present a curtailed version of the architecture that we use to analyze biology sequences and other health data in Figure 5.

|  |  |  |  |
| --- | --- | --- | --- |
|

|  |  |
| --- | --- |
| Processes:P1 – Pairwise distance calculationP2 – Multi-dimensional scalingP3 – Pairwise clustering (DAPWC)P4 – Visualization | Data:D1 – Input dataD2 – Distance matrixD3 – Three dimensional coordinatesD4 – Cluster mappingD5 – Plot file |

 | P1P2P3P4 |

Figure 5 Simple pipeline of steps

D1 and P1 in Figure 5 is a data and a process pair such that P1 can compute a pairwise distance matrix for items in D1 to produce D2. The process is general from D2 onwards, where MDS makes 3D projection of items in D2 while DAPWC (P3) finds clusters from D2. The 3D projection and cluster information are combined to form the final plot D5.

Our work on biology sequences uses sequence data, usually in FASTA [53] format, and an aligner such as Smith-Waterman [2] or Needleman-Wunsch [54] for D1 and P1. P1 is pleasingly parallel in problem architecture and we provide a Java implementation on Twister [43] and a C# flavor with MPI.NET [42]. We have similar distance computation implementations for vector data, where we have experimented with a couple of similarity measures based on Euclidean distance and Pearson correlation.

MDS run over D2 is fairly straightforward compared to clustering, which usually requires multiple sub runs to get the final clustering results. The reason for this is that the number of clusters required is not known in advance and hence we run it in a hierarchical fashion. For example, we start with finding a smaller number of clusters and re-cluster them further as necessary. The decision to further breakdown a cluster is guided with the aid of the 3D plot created with current clustering results and MDS mapping. We discuss this process and verification of results in [55, 56].

We perform other tasks such as cluster center finding and 3D phylogenetic tree creation [57] beyond what is shown in Figure 5. The largest dataset we have analyzed contains a half a million unique sequences [56] and we end this discussion with a few snapshots of 3D plots in Figure 6.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1. 100,000 fungi sequences
 | 1. 3D phylogenetic tree
 | 1. 3D plot of vector data
 |  |

Figure 6 Sample 3D plots

**Performance Results**

MDS and clustering Ogres were originally written in C# based on MPI.NET and targeting Windows HPC systems; however, in the attempts to make SPIDAL, we decided to move with Java for reasons 1) Java is dominant in the Big Data applications, frameworks, and libraries, so we could use or integrate SPIDAL with existing Big Data technologies 2) productivity offered by Java and its ecosystem, and 3) emerging success of Java in HPC [58]. We have done initial evaluations on SPIDAL for micro-benchmarks and full application benchmarks as described below.

**Systems and Software Information**

***Computer Systems***

We used two Indiana University clusters, Madrid and Tempest, and one FutureGrid [59] cluster – India, as described below.

* **Tempest:** 32 nodes, each has 4 Intel Xeon E7450 CPUs at 2.40GHz with 6 cores, totaling 24 cores per node; 48 GB node memory and 20Gbps Infiniband (IB) network connection. It runs Windows Server 2008 R2 HPC Edition – version 6.1 (Build 7601: Service Pack 1).
* **Madrid:** 8 nodes, each has 4 AMD Opteron 8356 at 2.30GHz with 4 cores, totaling 16 cores per node; 16GB node memory and 1Gbps Ethernet network connection. It runs Red Hat Enterprise Linux Server release 6.5
* **India cluster on FutureGrid (FG):** 128 nodes, each has 2 Intel Xeon X5550 CPUs at 2.66GHz with 4 cores, totaling 8 cores per node; 24GB node memory and 20Gbps IB network connection. It runs Red Hat Enterprise Linux Server release 5.10.

***Software Packages***

We used .NET 4.0 runtime and MPI.NET 1.0.0 for C# based tests. DA-VS and DA-PWC Java versions use a novel parallel tasks library called Habanero Java (HJ) library from Rice University [60, 61], which requires Java 8.

There have been several message passing frameworks for Java [62], but due to the lack of support for Infiniband (IB) network and to other drawbacks discussed in [63], we decided to evaluate OpenMPI with its Java binding and FastMPJ, which is a pure Java implementation of mpiJava 1.2 [64] specification. OpenMPI’s Java binding [65] is an adaptation from the original mpiJava library [66]. However, OpenMPI community has recently introduced major changes to its API, and internals, especially removing MPI.OBJECT type and adding support for direct buffers in Java. These changes happened while evaluating DA-VS and DA-PWC, thus they were tested with OpenMPI Java binding in one of its original (nightly snapshot version 1.9a1r28881) and updated forms (source tree revision 30301). Some tests were carried out with even newer versions – 1.7.5, 1.8, and 1.8.1. These are referred hereafter as OMPI-nightly, OMPI-trunk, OMPI-175, OMPI-18, and OMPI-181 for simplicity.

**Micro-benchmarks**

DA-VS and DA-PWC code rely heavily on a few MPI operations – allreduce, send and receive, broadcast, and allgather. We studied their Java performance against native implementations with micro-benchmarks adapted from OSU micro-benchmarks suite [7].

|  |  |
| --- | --- |
|  |  |
| 1. Performance of allreduce operation
 | 1. Performance of send/receive operations
 |
|  |  |
| 1. Performance of broadcast operation
 | 1. Performance of allgather operation
 |
|  |  |
| 1. Performance of allreduce on IB and Ethernet
 | 1. Performance of send/receive on IB and Ethernet
 |

Figure 7 MPI micro-benchmark performance

Figure 7 (a) shows the results of allreduce benchmark for different MPI implementations with message size ranging from 4 bytes (B) to 8 megabytes (MB). These are averaged values over patterns 1x1x8, 1x2x8, and 1x4x8 where pattern format is number of concurrent tasks (CT) per process x number of processes per node x number of nodes (i.e. TxPxN).

The best performance came with C versions of OpenMPI, but interestingly OMPI-trunk Java performance overlaps on these indicating its near zero overhead. The older, OMPI-nightly Java performance is near as well, but shows more overhead than its successor. Improvement of performance for latest OpenMPI versions is due to the use of direct buffers in Java, which are allocated outside of the garbage collected memory in Java virtual machine (JVM) and can be passed on to native operations as a reference without requiring an explicit copy. MPI send and receive, broadcast, and allgather performances followed a similar pattern as seen in Figure 7 (b), (c), and (d). Also note that FastMPJ though being a pure Java implementation came close to native performance in Figure 7 (b), (c), and (d); however, it did not perform well in the allreduce benchmark.

Figure 7 (c) and (d) show MPI allreduce, and send and receive performance with and without IB. While the decrease in communication times with IB is as expected, the near identical performance of Java with native benchmark in both IB and Ethernet cases is promising for our purpose.

**DA-PWC Benchmarks**

DA-PWC is a full application benchmark done with real data with 10k, 12k, 20k, and 40k points. Performance of DA-PWC changes with factors such as data size, number of clusters to find, number of iterations, and cooling parameters, thus all of these were fixed except data size throughout tests. The OpenMPI framework used was OMPI-181 and all testing were done in FG with IB. While the intention is to show how the Java based implementations in SPIDAL performs, results for C# version of DA-PWC are also included where available since most of the tests were carried out in parallel for comparison purposes. Note, SPIDAL does not offer a FastMPJ based DA-PWC implementation due to frequent runtime errors encountered in the initial tests.

***Load Test***

This evaluates DA-PWC for varying load against a fixed parallelism. We used 32 nodes each running with 8 way parallelism totaling 256 way parallelism.

|  |  |
| --- | --- |
|  |  |
| 1. Load test performance
 | 1. Load test performance as a ratio to 10k sample
 |

Figure 8 DA-PWC load test

DA-PWC is an $O\left(N^{2}\right)$ algorithm, thus it is usually expected for the runtime to follow a square relationship – or better, not to exceed that – with the number of points. We see from Figure 8 that while Java implementation handles the load increase gracefully, the C# fails to do so and deviates to a sudden increase in running time from 20k onwards.

***Strong Scaling Tests***

Strong scaling tests are a series of runs evaluating the performance of DA-PWC for varying number of nodes against fixed loads. The graphs in Figure 9 (a) to (i) present performance, speedup, and parallel efficiency for 40k, 20k, and 12k data. We define speedup and parallel efficiency against a base case that is not serial as shown in Equation 2 and Equation 3, where $p$ is the tested parallelism, $b$ is the base parallelism, $T\_{p}$ is the time taken with $p$ parallelism, and $T\_{b}$ is the time taken for the base case. This is necessary as larger data sets such as 20k and 40k could not be run serially to finish in a meaningful time. Also, these equations reduce to the usual ones when the base case parallelism is $1$ instead of $b$.

|  |  |
| --- | --- |
|  |  |
| 1. 40k performance
 | 1. 40k speedup
 |
|  |  |
| 1. 40k parallel efficiency
 | 1. 20k performance
 |
|  |  |
| 1. 20k speedup
 | 1. 20k parallel efficiency
 |
|  |  |
| 1. 12k performance
 | 1. 12k speedup
 |
|  |

|  |  |  |
| --- | --- | --- |
| $$Speedup \left(S\_{p}\right)=\frac{T\_{p}}{T\_{b}}$$ |

|  |
| --- |
|  |

Equation 2 |
| $$ Efficiency \left(E\_{p}\right)=b\*\frac{S\_{p}}{p}$$ |

|  |
| --- |
|  |

Equation 3 |

 |
| 1. 12k parallel efficiency
 |  |

Figure 9 DA-PWC strong scaling

A scalable solution is expected to show linear speedup ideally in a strong scaling test and Figure 9 (b), (e), and (h) verify DA-PWC Java version is scalable with near linear speedups. The C# version shows scalability for the smaller 12k sample, but does not scale well in the large 40k case. Another desirable property is to have parallel efficiencies around 1.0, which too is satisfied by the Java DA-PWC as seen in Figure 9 (c), (f), and (i). Again, C# version did well for the smaller case, but not for the 40k data set.

|  |  |
| --- | --- |
|  |  |
| 1. Strong scaling speedup for all data
 | 1. Strong scaling parallel efficiency for all data
 |

Figure 10 DA-PWC strong scaling results for all data sets

Figure 10 summarizes speedups and parallel efficiencies for all datasets across parallelisms from 1 to 256. These graphs are based on a serial base case that we obtained through extrapolating running times for large data on smaller parallelisms. The intention of this is to illustrate the behavior with increasing parallelism for different data sizes. It shows that DA-PWC scales well with the increase in parallelism within the shown limits for 20k and 40k data sets. The 12k data set shows reasonable scaling within 8 and 64 way parallelisms, but not outside this range. One reason for this behavior could be due to not having enough computation workload when 12k is split across 128 or 256 processes. This could lead to spending more time in communications causing a performance degrade.

1. Baru, C., M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl, *Setting the Direction for Big Data Benchmark Standards*, Chapter 14 in *Selected Topics in Performance Evaluation and Benchmarking*, R. Nambiar and M. Poess, Editors. 2013, Springer Berlin Heidelberg. p. 197-208. <http://dx.doi.org/10.1007/978-3-642-36727-4_14>.

2. Smith, T.F. and M.S. Waterman, *Identification of common molecular subsequences.* Journal of Molecular Biology, 1981. *147*(*1*): p. 195-197.

3. Nyberg, C., M. Shah, N. Govindaraju, and J. Gray. *Sort Benchmark Home Page*. Available from: <http://sortbenchmark.org/>.

4. Ruben S. Monteroa, Rafael Moreno-Vozmediano, and Ignacio M. Llorente, *An elasticity model for High Throughput Computing clusters.* Journal of Parallel and Distributed Computing, 24 May, 2010. DOI:<http://dx.doi.org/10.1016/j.jpdc.2010.05.005>

5. Ghazal, A., T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, *BigBench: towards an industry standard benchmark for big data analytics*, in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013, ACM. New York, New York, USA. pages. 1197-1208. DOI: 10.1145/2463676.2463712.

6. Baru, C. and T. Rabl, *Big Data Benchmarking*. 2014. <http://cci.drexel.edu/bigdata/bigdata2014/IEEE2014TutorialBaruRabl.pdf>.

7. Laboratory, T.O.S.U.s.N.-B.C. and (NBCL). *OSU Micro-Benchmarks*. Available from: <http://mvapich.cse.ohio-state.edu/benchmarks/>.

8. Huang, S., J. Huang, J. Dai, T. Xie, and B. Huang, *The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis*, Chapter 9 in *New Frontiers in Information and Software as Services*, D. Agrawal, K.S. Candan, and W.-S. Li, Editors. 2011, Springer Berlin Heidelberg. p. 209-228. <http://dx.doi.org/10.1007/978-3-642-19294-4_9>.

9. Bader, D.A., J. Berry, S. Kahan, R. Murphy, E.J. Riedy, and J.W.I. University), *Graph 500 Benchmark*. 2011. <http://www.graph500.org/specifications>.

10. Dongarra, J., *The LINPACK Benchmark: An Explanation*, in *Proceedings of the 1st International Conference on Supercomputing*. 1988, Springer-Verlag. pages. 456-474.

11. Anderson, E., Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J.D. Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen, *LAPACK: a portable linear algebra library for high-performance computers*, in *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*. 1990, IEEE Computer Society Press. New York, New York, USA. pages. 2-11.

12. Blackford, L.S., J. Choi, A. Cleary, A. Petitet, R.C. Whaley, J. Demmel, I. Dhillon, K. Stanley, J. Dongarra, S. Hammarling, G. Henry, and D. Walker, *ScaLAPACK: a portable linear algebra library for distributed memory computers - design issues and performance*, in *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*. 1996, IEEE Computer Society. Pittsburgh, Pennsylvania, USA. pages. 5. DOI: 10.1145/369028.369038.

13. Dongarra, J., E. Strohmaier, and a.M. Resch. *Top 500 Supercomputing Sites*. 1/2/2015]; Available from: <http://www.top500.org/>.

14. Petitet, A., R.C. Whaley, J. Dongarra, and A. Cleary. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. 2008 9/18/2008 1/2/2015]; Available from: <http://www.netlib.org/benchmark/hpl/>.

15. D'Azevedo, E., K. Wong, P.H. Che, C.S. Lee, T. Chan, R. Wong, and R. Barrett. *Performance Results*. mHPL: Modified HPL Project 1/2/2015]; Available from: <http://www.nics.tennessee.edu/sites/default/files/HPL-site/performance.html>.

16. Dongarra, J.J. *Performance of Various Computers Using Standard Linear Equations Software*. 2014 June 15, 2014 1/2/2015]; Available from: <http://www.netlib.org/benchmark/performance.ps>.

17. Petitet, A., R.C. Whaley, J. Dongarra, and A. Cleary. *HPL Algorithm*. 2008 [accessed 2015 1/2/2015]; Available from: <http://www.netlib.org/benchmark/hpl/algorithm.html>.

18. Dongarra, J.J., P. Luszczek, and A. Petitet, *The LINPACK Benchmark: past, present and future.* Concurrency and Computation: Practice and Experience, 2003. 15(9): p. 803-820. DOI:10.1002/cpe.728. <http://dx.doi.org/10.1002/cpe.728>

19. Frumkin, M.A., M. Schultz, H. Jin, and J. Yan, *Implementation of the NAS Parallel Benchmarks in Java*. <http://www.nas.nasa.gov/assets/pdf/techreports/2002/nas-02-009.pdf>.

20. *OpenMP API specification for parallel programming*. [accessed 2010 November 26]; Available from: <http://openmp.org/wp/>.

21. Shantenu Jha, Andre Merzky, and Geoffrey Fox, *Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes.* Concurrency and Computation: Practice and Experience, 2009. 21(8): p. 1087-1108. DOI:<http://dx.doi.org/10.1002/cpe.1406>. <http://grids.ucs.indiana.edu/ptliupages/publications/cloud-grid-saga_rev.pdf>

22. Lei, W., Z. Jianfeng, L. Chunjie, Z. Yuqing, Y. Qiang, H. Yongqiang, G. Wanling, J. Zhen, S. Yingjie, Z. Shujie, Z. Chen, L. Gang, K. Zhan, L. Xiaona, and Q. Bizhu. *BigDataBench: A big data benchmark suite from internet services*. in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. 15-19 Feb. 2014 2014.

23. Luo, C., W. Gao, Z. Jia, R. Han, J. Li, X. Lin, L. Wang, Y. Zhu, and J. Zhan. *Handbook of BigDataBench (Version 3.1) - A Big Data Benchmark Suite*. Available from: <http://prof.ict.ac.cn/BigDataBench/wp-content/uploads/2014/12/BigDataBench-handbook-6-12-16.pdf>.

24. Kamburugamuve, S. *Survey of Apache Big Data Stack* December 19 2013 Available from: <http://grids.ucs.indiana.edu/ptliupages/publications/survey_apache_big_data_stack.pdf>.

25. Qiu, J., G. Fox, and S. Jha. *Kaleidoscope of (Apache) Big Data Stack (ABDS) and HPC Technologies* December 24 2014 Available from: <http://grids.ucs.indiana.edu/ptliupages/publications/KaleidoscopeTable2.docx>.

26. Jolliffe, I., *Principal Component Analysis*, Chapter in *Encyclopedia of Statistics in Behavioral Science*. 2005, John Wiley & Sons, Ltd. <http://dx.doi.org/10.1002/0470013192.bsa501>.

27. Glotzer, S.C., B. Panoff, and S. Lathrop, *Challenges and Opportunities in Preparing Students for Petascale Computational Science and Engineering.* Computing in Science & Engineering, September, 2009. 11(5): p. 22-27. DOI:10.1109/MCSE.2009.134. <http://portal.acm.org/citation.cfm?id=1608634>

28. Paulson, E. *HDFS Benchmarks*. Available from: <http://epaulson.github.io/HadoopInternals/benchmarks.html#dfsio>.

29. Shodor Foundation. *Home page*. 2009 [accessed 2009 December]; Available from: <http://www.shodor.org/>.

30. Wang, D., M. Shi, G. Huang, and J. Duan. *HiBench*. Available from: <https://github.com/intel-hadoop/HiBench>.

31. *The Graph 500 List (November 2014)*. Available from: <http://www.graph500.org/>.

32. Gowda, B.D. and N. Ravi. *BigBench: Toward An Industry-Standard Benchmark for Big Data Analytics*. 2014 November 25, 2014 1/4/15]; Available from: <http://blog.cloudera.com/blog/2014/11/bigbench-toward-an-industry-standard-benchmark-for-big-data-analytics/>.

33. Rabl, T., M. Poess, C. Baru, and H.A. Jacobsen, *Specifying Big Data Benchmarks: First Workshop, WBDB 2012, San Jose, CA, USA, May 8-9, 2012 and Second Workshop, WBDB 2012, Pune, India, December 17-18, 2012, Revised Selected Papers*. 2014: Springer Berlin Heidelberg. <http://books.google.com/books?id=49GhngEACAAJ>. ISBN:9783642539732

34. Manyika, J., M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A.H. Byers, *Big data: The next frontier for innovation, competition, and productivity*. May 2011, 2011. <http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation>.

35. Frank, M., M. Poess, and T. Rabl, *Efficient update data generation for DBMS benchmarks*, in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. 2012, ACM. Boston, Massachusetts, USA. pages. 169-180. DOI: 10.1145/2188286.2188315.

36. *Language Manual*. 2014 Oct 22, 2014 1/4/2015]; Available from: <https://cwiki.apache.org/confluence/display/Hive/LanguageManual>.

37. Armstrong, T.G., V. Ponnekanti, D. Borthakur, and M. Callaghan, *LinkBench: a database benchmark based on the Facebook social graph*, in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013, ACM. New York, New York, USA. pages. 1185-1196. DOI: 10.1145/2463676.2465296.

38. Narayanan, R., B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. *MineBench: A Benchmark Suite for Data Mining Workloads*. in *Workload Characterization, 2006 IEEE International Symposium on*. 25-27 Oct. 2006 2006.

39. Zambreno, J., B.Ö. Ikyılmaz, G. Memik, and A. Choudhary, *Performance characterization of data mining applications using MineBench (2006)*, in *9th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*. 2006. [www.bioperf.org/ZOM06.pdf](http://www.bioperf.org/ZOM06.pdf).

40. Ozisikyilmaz, B., R. Narayanan, J. Zambreno, G. Memik, and A. Choudhary. *An Architectural Characterization Study of Data Mining and Bioinformatics Workloads*. in *Workload Characterization, 2006 IEEE International Symposium on*. 25-27 Oct. 2006 2006.

41. Barahmand, S. and S. Ghandeharizadeh, *BG: A Benchmark to Evaluate Interactive Social Networking Actions*, in *CIDR*. 2013, [www.cidrdb.org](http://www.cidrdb.org). <http://dblp.uni-trier.de/db/conf/cidr/cidr2013.html#BarahmandG13>.

42. Open Sysem Lab, I.U.B. *MPI.NET*. 2008 Available from: <http://osl.iu.edu/research/mpi.net/>.

43. J.Ekanayake, H.Li, B.Zhang, T.Gunarathne, S.Bae, J.Qiu, and G.Fox, *Twister: A Runtime for iterative MapReduce*, in *Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010*. 2010, ACM. Chicago, Illinois. <http://grids.ucs.indiana.edu/ptliupages/publications/hpdc-camera-ready-submission.pdf>.

44. Zhang, B., Y. Ruan, and J. Qiu, *Harp: Collective Communication on Hadoop*. 2014. <http://grids.ucs.indiana.edu/ptliupages/publications/HarpQiuZhang.pdf>.

45. Zaharia, M., M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica, *Spark: cluster computing with working sets*, in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. 2010, USENIX Association. Boston, MA. pages. 10-10.

46. Vega-Gisbert, O. and J.E. Roman, *Design and implementation of Java bindings in Open MPI*. 2014. users.dsic.upv.es/~jroman/preprints/ompi-java.pdf.

47. Gill, P.E., W. Murray, and M.H. Wright, *Practical optimization.* International Journal for Numerical Methods in Engineering, 1982. 18(6): p. 954-954. DOI:10.1002/nme.1620180612. <http://dx.doi.org/10.1002/nme.1620180612>

48. Ruan, Y. and G. Fox, *A Robust and Scalable Solution for Interpolative Multidimensional Scaling with Weighting*, in *Proceedings of the 2013 IEEE 9th International Conference on e-Science*. 2013, IEEE Computer Society. pages. 61-69. DOI: 10.1109/eScience.2013.30.

49. Rose, K., E. Gurewwitz, and G. Fox, *A deterministic annealing approach to clustering.* Pattern Recogn. Lett., 1990. 11(9): p. 589-594. DOI:10.1016/0167-8655(90)90010-y

50. Fox, G.C. and S. Ekanayake. *Deterministic Annealing Pairwise Clustering*. Available from: <https://github.com/DSC-SPIDAL/dapwc>.

51. Fox, G., D.R. Mani, and S. Pyne, *Parallel deterministic annealing clustering and its application to LC-MS data analysis*, in *BigData Conference*. 2013, IEEE. pages. 665-673.

52. Fox, G.C. and S. Ekanayake. *Deterministic Annealing Vector Sponge*. Available from: <https://github.com/DSC-SPIDAL/davs>.

53. Sector/Sphere. *High Performance Distributed File System and Parallel Data Processing Engine*. [accessed 2010 November 7]; Available from: <http://sector.sourceforge.net>.

54. Needleman, S.B. and C.D. Wunsch, *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* J Mol Biol, Mar, 1970. 48(3): p. 443-53. <http://www.ncbi.nlm.nih.gov/pubmed/5420325>

55. Ekanayake, S. *Study of Biological Sequence Clustering*. 2013 Available from: <http://grids.ucs.indiana.edu/ptliupages/publications/study_of_sequence_clustering_formatted_v2.pdf>.

56. Ekanayake, S., Y. Ruan, and G.C. Fox. *Million Sequence Clustering*. Available from: <http://salsahpc.indiana.edu/millionseq/>.

57. Yang Ruan, G.L.H., Saliya Ekanayake, Ursel Schütte, James D. Bever, Haixu Tang, Geoffrey Fox, *Integration of Clustering and Multidimensional Scaling to Determine Phylogenetic Trees as Spherical Phylograms Visualized in 3 Dimensions*, in *C4Bio 2014 of IEEE/ACM CCGrid 2014*. May 26-29, 2014, 2014. Chicago, USA. <http://salsahpc.indiana.edu/millionseq/fungi2_phylo/reference/Integration%20of%20Clustering%20and%20Multidimensional%20Scaling%20to%20Determine%20Phylogenetic%20Trees%20as%20Spherical%20Phylograms%20Visualized%20in%203%20Dimensions.pdf>.

58. Taboada, G.L., J. Touri, #241, Ram, #243, and n. Doallo, *Java for high performance computing: assessment of current research and practice*, in *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*. 2009, ACM. Calgary, Alberta, Canada. pages. 30-39. DOI: 10.1145/1596655.1596661.

59. von Laszewski, G., G.C. Fox, W. Fugang, A.J. Younge, A. Kulshrestha, G.G. Pike, W. Smith, Vo, x, J. ckler, R.J. Figueiredo, J. Fortes, and K. Keahey. *Design of the FutureGrid experiment management framework*. in *Gateway Computing Environments Workshop (GCE), 2010*. 14-14 Nov. 2010 2010.

60. Cav, V., #233, J. Zhao, J. Shirako, and V. Sarkar, *Habanero-Java: the new adventures of old X10*, in *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java*. 2011, ACM. Kongens Lyngby, Denmark. pages. 51-61. DOI: 10.1145/2093157.2093165.

61. Sarkar, V.a.I., Shams Mahmood *HJ Library*. Available from: [https://wiki.rice.edu/confluence/display/PARPROG/HJ+Library](https://wiki.rice.edu/confluence/display/PARPROG/HJ%2BLibrary).

62. Taboada, G.L., S. Ramos, R.R. Exp, #243, sito, J. Touri, #241, Ram, #243, and n. Doallo, *Java in the High Performance Computing arena: Research, practice and experience.* Sci. Comput. Program., 2013. 78(5): p. 425-444. DOI:10.1016/j.scico.2011.06.002

63. Expósito, R., S. Ramos, G. Taboada, J. Touriño, and R. Doallo, *FastMPJ: a scalable and efficient Java message-passing library.* Cluster Computing, 2014/02/06, 2014: p. 1-20. DOI:10.1007/s10586-014-0345-4. <http://dx.doi.org/10.1007/s10586-014-0345-4>

64. Bryan Carpenter, G.F., Sung-Hoon Ko and Sang Lim, *mpiJava 1.2: API Specification*. 1999. <https://www.open-mpi.org/papers/mpi-java-spec/mpiJava-spec.pdf>.

65. Oscar Vega-Gisbert, J.E.R., Jeffrey M. Squyres, *Design and implementation of Java bindings in Open MPI*. 2014. users.dsic.upv.es/~jroman/preprints/ompi-java.pdf.

66. Baker, M., B. Carpenter, G. Fox, S. Hoon Ko, and S. Lim, *mpiJava: An object-oriented java interface to MPI*, Chapter 82 in *Parallel and Distributed Processing*, J. Rolim, F. Mueller, A. Zomaya, F. Ercal, S. Olariu, B. Ravindran, J. Gustafsson, H. Takada, R. Olsson, L. Kale, P. Beckman, M. Haines, H. ElGindy, D. Caromel, S. Chaumette, G. Fox, Y. Pan, K. Li, T. Yang, G. Chiola, G. Conte, L.V. Mancini, D. Méry, B. Sanders, D. Bhatt, and V. Prasanna, Editors. 1999, Springer Berlin Heidelberg. p. 748-762. <http://dx.doi.org/10.1007/BFb0097964>.