Portable Parallel Programming on Cloud and HPC: Scientific Applications of Twister4Azure

Thilina Gunarathne, Bingjing Zhang, Tak-Lon Wu, Judy Qiu
School of Informatics and Computing
Indiana University, Bloomington.
{tgunarat, zhangbj, taklwu, xqiu}@indiana.edu

Abstract— **Recent advancements in data intensive computing for science discovery are fueling a dramatic growth in use of data-intensive iterative computations. The utility computing model introduced by cloud computing combined with the rich set of cloud infrastructure services offers a very attractive environment for scientists to perform such data intensive computations. The challenges to large scale distributed computations on *clouds* demand new computation frameworks that are specifically tailored for cloud characteristics in order to easily and effectively harness the power of *clouds*. Twister4Azure is a distributed decentralized iterative MapReduce runtime for Windows Azure Cloud. It extends the familiar, easy-to-use MapReduce programming model with iterative extensions, enabling a wide array of large-scale iterative data analysis for scientific applications on Azure cloud. This paper presents the applicability of Twister4Azure with highlighted features of fault-tolerance, efficiency and simplicity. We study three data-intensive applications − two iterative scientific applications, Multi-Dimensional Scaling and KMeans Clustering; one data–intensive pleasingly parallel scientific application, BLAST+ sequence searching. Performance measurements show comparable or a factor of 2 to 4 better results than the traditional MapReduce runtimes deployed on up to 256 instances and for jobs with tens of thousands of tasks.**

Keywords- Iterative MapReduce, Cloud Computing, HPC, Scientific applications

I. INTRODUCTION

The current scientific computing landscape is vastly populated by the growing set of data-intensive computations that require enormous amounts of computational as well as storage resources and novel distributed computing frameworks. The pay-as-you-go Cloud computing model provides an option for the computational and storage needs of such computations. The new generation of distributed computing frameworks such as MapReduce focuses on catering to the needs of such data-intensive computations.

Iterative computations are at the core of the vast majority of scientific computations. Many important data intensive iterative scientific computations can be implemented as iterative computation and communication steps, in which computations inside an iteration are independent and are synchronized at the end of each iteration through reduce and communication steps, making it possible for individual iterations to be parallelized using technologies such as MapReduce. Examples of such applications include dimensional scaling, many clustering algorithms, many machine learning algorithms, and expectation maximization applications, among others. The growth of such data intensive iterative computations in number as well as importance is driven partly by the need to process massive amounts of data and partly by the emergence of data intensive computational fields, such as bioinformatics, chemical informatics and web mining.

Twister4Azure is a distributed decentralized iterative MapReduce runtime for Windows Azure Cloud that was developed utilizing Azure cloud infrastructure services. Twister4Azure extends the familiar, easy-to-use MapReduce programming model with iterative extensions, enabling a wide array of large-scale iterative data analysis and scientific applications to utilize Azure platform easily and efficiently in a fault-tolerant manner. Twister4Azure effectively utilizes the eventually-consistent, high-latency Azure cloud services to deliver performance that is comparable to traditional MapReduce runtimes for non-iterative MapReduce. It outperforms traditional MapReduce runtimes for iterative MapReduce computation. Twister4Azure has minimal management & maintenance overheads and provides users with the capability to dynamically scale up or down the amount of computing resources. Twister4Azure takes care of almost all the Azure infrastructure (service failures, load balancing, etc) and coordination challenges, and frees users from having to deal with cloud services. Window Azure claims to allow the users to "Focus on your applications, not the infrastructure." Twister4Azure take it one step further and lets users focus only on the application logic without worrying about the application architecture.

Applications of Twister4Azure can be categorized as three classes of application patterns. First are the Map only applications, which are also called pleasingly (or embarrassingly) parallel applications. Example of this type of applications include Monte Carlo simulations, BLAST+ sequence searches, parametric studies and most of the data cleansing and pre-processing applications. Section VI analyzes the BLAST+[1] Twister4Azure application.

The second type of applications includes the traditional MapReduce type applications, which utilize the reduction phase and other features of MapReduce. Twister4Azure contains sample implementations of SmithWatermann-GOTOH (SWG)[2] pairwise sequence alignment and Word Count as traditional MapReduce type applications.

The third and most important type of applications Twister4Azure supports is the iterative MapReduce type applications. As mentioned above, there exist many data-intensive scientific computation algorithms that rely on iterative computations, wherein each iterative step can be easily specified as a MapReduce computation. Section IV

and V present detailed analysis of Kmeans Clustering and MDS iterative MapReduce implementations. Twister4Azure also contains an iterative MapReduce implementation of PageRank and we are actively working on implementing more iterative scientific applications using Twister4Azure.

Developing Twister4Azure was an incremental process, which began with the development of pleasingly parallel cloud programming frameworks[3] for bioinformatics applications utilizing cloud infrastructure services. MRRoles4Azure[4] MapReduce framework for Azure cloud was developed based on the success of pleasingly parallel cloud frameworks and was released in December 2010. We started working on Twister4Azure to fill the void of distributed parallel programming frameworks in the Azure environment (as of June 2010) and the first public beta release of Twister4Azure[5] occurred in May 2011.

II. BACKGROUND

A. MapReduce

The MapReduce[6] data-intensive distributed computing paradigm was introduced by Google as a solution for processing massive amounts of data using commodity clusters. MapReduce provides an easy-to-use programming model that features fault tolerance, automatic parallelization, scalability and data locality-based optimizations. Apache Hadoop[7] MapReduce is a widely used open-source implementation of the Google MapReduce distributed data processing framework.

B. Twister

The Twister[8] iterative MapReduce framework is an expansion of the traditional MapReduce programming model, which supports traditional as well as iterative MapReduce data-intensive computations. Twister supports MapReduce in the manner of "configure once, and run many time". During the configuration stage, static data is configured and loaded into Map or Reduce tasks, and then reused through the iterations. In each iteration, the data is first mapped in the compute nodes, and reduced, then combined back to the driver node (control node). With these features, Twister supports iterative MapReduce computations efficiently when compared to other traditional MapReduce runtimes such as Hadoop[9]. Fault detection and recovery are also supported between the iterations. In this paper we use the java implementation of Twister and identify it as Java HPCTwister.

Java HPCTwister has one driver node for controlling and the *Map* and *Reduce* tasks are implemented as working threads managed by daemon process on each worker node. Daemons communicate with the driver node and with each other through messages. For command, communication and data transfers, Twister uses a Publish/Subscribe messaging middleware system and ActiveMQ[10] is used for the current experiments. Twister does not currently have an integrated distributed file system and the data distribution and management are operated through scripts.

C. Microsoft Azure platform

The Microsoft Azure platform [16] is a cloud computing platform that offers a set of cloud computing services. Windows Azure Compute allows the users to lease Windows virtual machine instances and offers the .net runtime as the platform through two programmable roles called Worker Roles and Web Roles. Starting recently Azure also supports VM roles (beta), giving the ability for users to directly deploy virtual machine instances. Azure offers a limited set of instances on a linear price and feature scale[11]. Azure small instance contains one 1.6GHz CPU core with 1.75GB memory and costs 0.12$ per hour. Medium, Large and Extra Large instances multiply the features and the cost of small instances by a factor of 2, 4 and 8 respectively.

The Azure Storage Queue is an eventual consistent, reliable, scalable and distributed web-scale message queue service that is ideal for small, short-lived, transient messages. The Azure queue does not guarantee the order of the messages, the deletion of messages or the availability of all the messages for a single request, although it guarantees eventual availability over multiple requests. Each message has a configurable visibility timeout. Once it is read by a client, the message will not be visible for other clients until the visibility time expires or if the previous reader delete it.

The Azure Storage Table service offers a large-scale eventually consistent structured storage. Azure Table can contain a virtually unlimited number of entities (*aca* records or rows) that can be up to 1MB. Entities contain properties (*aca* cells), that can be up to 64KB. A table can be partitioned to store across many nodes for scalability.

The Azure Storage BLOB service provides a web-scale distributed storage service in which users can store and retrieve any type of data through a web services interface.

D. MRRoles4Azure

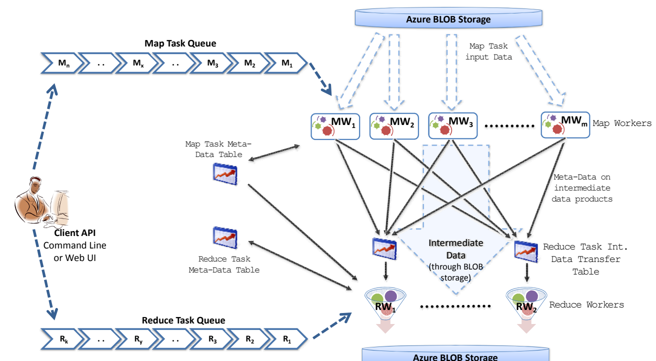

Figure 1. MRRoles4Azure Architecture[1]

MRRoles4Azure is a distributed decentralized MapReduce runtime for Windows Azure cloud platform that utilizes Azure cloud infrastructure services. MRRoles4Azure overcomes the latencies of cloud services by using sufficiently coarser grained map and reduce tasks. It overcomes the eventual data availability of cloud storage services through re-trying and explicitly designing the system so that it does not rely on the immediate availability of data across all distributed workers. As in Figure 1

MRRoles4Azure uses Azure Queues for map and reduce task scheduling, Azure Tables for metadata storage and monitoring data storage, Azure BLOB storage for data storage (input, output and intermediate) and the Window Azure Compute worker roles to perform the computations.

In order to withstand the brittleness of cloud infrastructures and to avoid potential single point failures, MR4Azure was designed as a decentralized control architecture which does not rely on a central coordinator or a client side driver. MR4Azure provides users with the capability to dynamically scale up/down the number of computing resources. The *Map* and *Reduce* tasks of the MR4Azure runtime are dynamically scheduled using global queues achieving natural load balancing given sufficient amount of tasks. MR4Azure handles task failures and slower tasks through re-execution and duplicate executions respectively. MapReduce architecture requires the reduce tasks to ensure the receipt of all the intermediate data products from Map tasks before beginning the reduce phase. Since ensuring such a collective decision is not possible with the direct use of eventual consistent tables, MRRoles4Azure uses additional data structures on top of Azure Tables for this purpose. Gunarathne et al.[1] presents more detailed description about MRRoles4Azure and show that MRRoles4Azure performs comparably to the other popular MapReduce runtimes.

III. TWISTER4AZURE – ITERATIVE MAPREDUCE

Twister4Azure extends the MRRoles4Azure to support iterative MapReduce executions, enabling a wide array of large-scale iterative data analysis and scientific applications to easily and efficiently utilize the Azure cloud platform in a fault-tolerant manner. Twister4Azure utilizes the scalable, distributed and highly-available Azure cloud services as the underlying building blocks and employs a decentralized control architecture avoiding single point failures.

A. *Twister4Azure Programming model*

There exists a significant amount of data analysis as well as scientific computation algorithms that rely on iterative computations, where each iterative step can easily be specified as a MapReduce computation. Typical data-intensive iterative computations follow the structure given in Code 1. We can identify two main types of data in these computations, the loop invariant input data and the loop variant delta values. Delta values are the result or a representation of the result of processing the input data in each iteration. These delta values are used in the computation of the next iteration. One example of such delta values would be the centroids in a KMeans Clustering computation (section IV). Single iterations of such computations are easy to parallelize by processing the data points or blocks of data points independently in parallel and performing synchronization between the iterations through communication steps.

Typical data-intensive iterative computations can be easily parallelized using the Twister4Azure iterative map reduce model. Twister4Azure will generate *map* tasks for each data block (line 5-7 in Code 1) and each *map* task will calculate a partial result, which will be communicated to the respective *reduce* tasks. The typical number of *reduce* tasks will be orders of magnitude less than the number of map tasks. Reduce tasks (line 8) will perform any necessary computations, combine the partial results received and output part of the final result. A single *merge* task will merge the results emitted by the *reduce* tasks and evaluate the loop conditional function (line 8 and line 4), often comparing the new delta results with the older delta results. The new delta output of the merge tasks will then be broadcasted to all the map tasks in the next iteration. Figure 2 presents the flow of the Twister4Azure programming model.

Code 1 Typical data-intensive iterative computation

```
1:    k ← 0;
2:    MAX ← maximum iterations
3:    δ[0] ← initial delta value
4:    while ( k< MAX_ITER || f(δ[k], δ[k-1]) )
5:        foreach datum in data
6:            β[datum] ← process (datum, δ[k])
7:        end foreach
8:        δ[k+1] ← combine(β[])
9:        k ← k+1
10:   end while
```

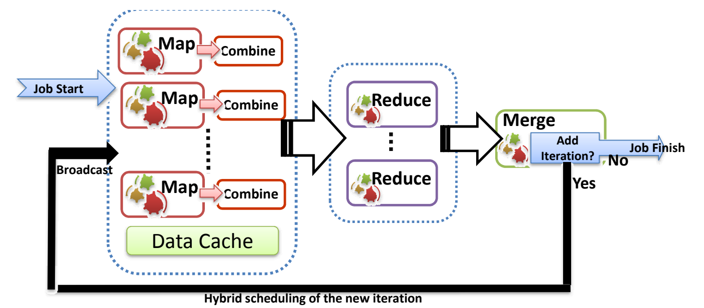

Figure 2. Twister4Azure programming model

1) Map and Reduce API

Twister4Azure extends the *map* and *reduce* functions of traditional MapReduce to include the broadcast data (delta values) as an input parameter. The broadcast data is provided as follows to the *Map* and *Reduce* task as a list of key-value pairs.

```
Map(<key>, <value>, list_of <key,value>)
Reduce(<key>, list_of <value>, list_of <key,value>)
```

2) Merge

Twister4Azure introduces *Merge* as a new step to the MapReduce programming model to support iterative applications; it executes after the *Reduce* step. *Merge* Task receives all the *Reduce* outputs and the broadcast data for the current iteration as the inputs. There can only be one *merge* task for a MapReduce job. With *merge*, the overall flow of the iterative MapReduce computation flow would look as follows.

Map -> Combine -> Shuffle -> Sort -> Reduce -> Merge

Since Twister4Azure does not have a centralized driver to take control decisions, the *Merge* step serves as the "loop-test" in the Twister4Azure decentralized architecture. Users can add a new iteration, finish the job or schedule a new MapReduce job from the *Merge* task. These decisions can be

made based on the number of iterations or on comparisons of the results from the previous iteration and the current iteration, such as the k-value difference between iterations for KMeansClustering. Users can use the results of the current iteration and the broadcast data to make these decisions. It is possible to specify the output of merge task as the broadcast data of the next iteration.

```
Merge(list_of <key,list_of<value>>,list_of <key,value>)
```

B. Data Cache

Twister4Azure In-Memory DataCache caches the loop-invariant (static) data across iterations in the memory of worker roles. Data caching avoids the download, loading and parsing cost of loop invariant input data, which gets reused in the iterations. These data products are comparatively larger sized and consist of traditional MapReduce key-value pairs. Twister4Azure maintains a single in-memory data cache storage per worker-role shared across *map*, *reduce* and *merge* workers, allowing the reuse of cached data across different tasks as well as across any MapReduce application within the same job. The caching of loop-invariant data gives significant speedups for the data-intensive iterative MapReduce applications. Broadcast data also utilize the data cache to optimize the data broadcasting as mentioned in Subsection D.

Twister4Azure also supports disk-based caching of the Azure Blobs. Twister4Azure stores all the files it downloads from the Blob storage in the local instance storage. Any request for a previously downloaded data product will be served from the local disk cache.

C. Cache Aware Scheduling

In order to take maximum advantage of the data caching for iterative computations, *Map* tasks of the subsequent iterations need to be scheduled with awareness of the data products cached in worker-roles. If the loop-invariant data for a *map* task is present in the DataCache of a certain worker-role then that map tasks should be scheduled to that particular worker-role. Decentralized architecture of Twister4Azure presents a challenge in this situation as Twister4Azure does not have a central entity which has a global view of the data products cached in the worker-roles or has the ability to push the tasks to a specific worker-role.

As a solution to the above issue, Twister4Azure opted for a model in which the workers pick tasks to execute based on the data products they have in their DataCache and based on the information that is published in to a central bulletin board (an Azure table). Naïve implementation of this model requires all the tasks for a particular job to be advertised, making the bulletin board a bottleneck. We avoid this by locally storing the executed *map* task execution histories (meta-data required for execution of a map task) for the cached data products. This allows the workers to start the execution of the map tasks for new iteration immediately after the workers get the information about a new iteration. With this optimization, the bulletin board only advertises information about the new iterations. As shown in Figure 3, new MapReduce jobs (non-iterative and 1st iteration of iterative) are scheduled through Azure queues.

Any tasks for an iteration that did not get scheduled in the above manner will be added back to the task scheduling queue by the first available worker without a matching task for execution. This ensures the eventual completion of the job and the fault tolerance of the tasks in the event of a worker failure and also ensures the dynamic scalability of the system when new workers are brought up. This mechanism can also be used to avoid the slow executing tail tasks of the iteration by duplicate execution in available instances. However, handling of slow executing tasks of iterations is still under development and is not used in the experiments that were performed for this paper.

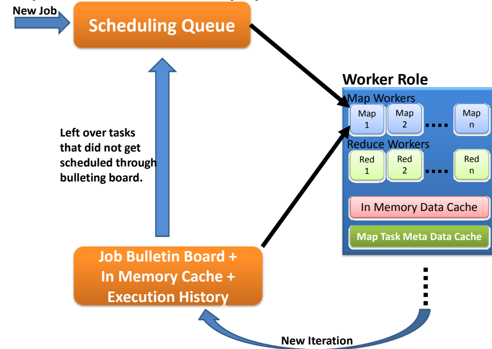

Figure 3. Cache Aware Hybrid Scheduling

D. Data broadcasting

The loop variant data (δ values in Code 1) needs to broadcasted to all the tasks in an iteration. With Twister4Azure users can specify broadcast data for iterative as well as non-iterative jobs. In typical data-intensive iterative computations, the loop-variant data (δ) is orders of magnitude smaller than the loop-invariant data. Currently Twister4Azure uses the Azure blob storage to communicate the broadcast data. Twister4Azure supports caching of broadcast data ensuring that only a single retrieval of Broadcast data occurs per node per iteration. This increases the efficiency of broadcasting when there are more than one *map/reduce/merge* worker per worker-role and when there are multiple waves of *map* tasks per iteration. Some of our experiments had more than 16 such tasks per worker-role.

E. Intermediate data communication

MRRoles4Azure uses the Azure blob storage to store intermediate data products and the Azure tables to store meta-data about intermediate data products, which performed well for non-iterative applications. Based on our experience, tasks in iterative MapReduce jobs are of relatively finer granular making the intermediate data communication overhead more prominent. They produce a large number of smaller intermediate data products causing the Blob storage based intermediate data transfer model to under-perform. Hence, we opted for a hybrid model, in which smaller data products are transferred through the Azure tables. Twister4Azure uses the intermediate data product meta-data table entry itself to store the intermediate data products up to a certain size (currently 64kb which is the limit for a single item in an Azure table entry) and use the blob storage for the data products that are larger than that limit. Additionally in

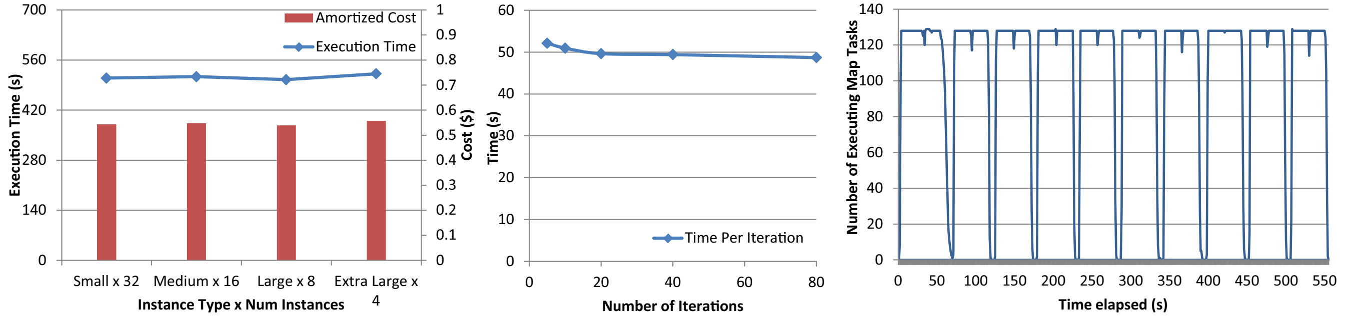

Figure 4. Twister4Azure KMeansClustering (20D data with 500 centroids, 32 cores). **Left(a)**: Instance type study with 10 iterations 32 million data points **Center(b)** : Time per iteration with increasing number of iterations 32 million data points. **Right(c):** Twister4Azure executing Map Task histogram for 128 million data points in 128 Azure small instances

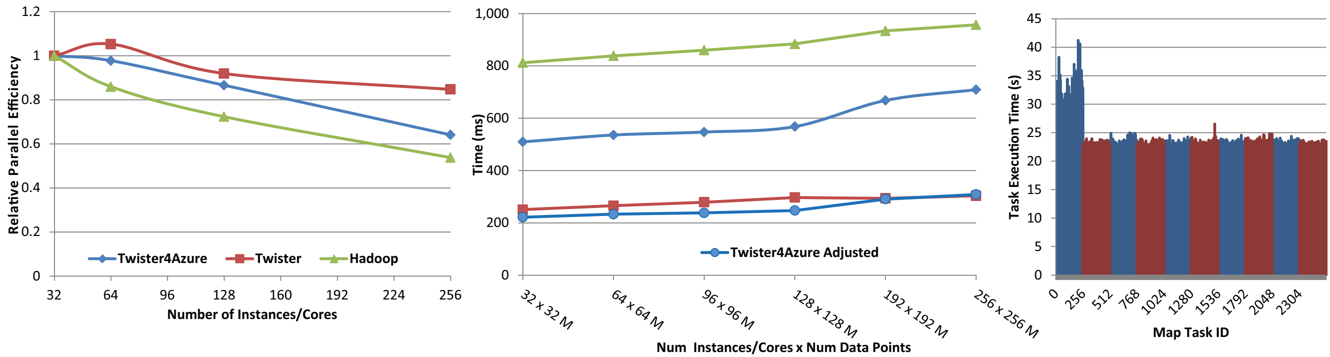

Figure 5. KMeansClustering Scalability. **Left(a):** Relative parallel efficiency of strong scaling using 128 million data points. **Center(b):** Weak scaling. Workload per core is kept constant (ideal is a straight horizontal line). **Right(c)** :Twister4Azure Map task execution time histogram for 128 million data points in 128 Azure small instances

Twister4Azure, all data communication is performed using asynchronous operations.

F. *Other features*

Twister4Azure supports typical MapReduce fault tolerance through re-execution of failed tasks, ensuring the eventual completion of the iterative computations.

Twister4Azure also supports the deployment of multiple MapReduce applications in a single deployment, making it possible to utilize more than one MapReduce application inside an iteration of a single job. This also enables workflow scenarios without redeployment. Twister4Azure also provides users with a web-based monitoring console from which they can monitor the progress of their jobs.

IV. KMeans Clustering

Clustering is the process of partitioning a given data set into disjoint clusters. The use of clustering and other data mining techniques to interpret very large data sets has become increasingly popular, with petabytes of data becoming commonplace. The K-Means clustering[12] algorithm has been widely used in many scientific and industrial application areas due to its simplicity and applicability to large data sets. We are currently working on a scientific project that requires clustering of several TeraBytes of data using KMeansClustering and millions of centroids.

K-Means clustering is often implemented using an iterative refinement technique, in which the algorithm iterates until the difference between cluster centers in subsequent iterations, i.e. the *error*, falls below a predetermined threshold. Each iteration performs two main steps, the cluster *assignment step*, and the centroids *update step*. In the MapReduce implementation, *assignment step* is performed in the Map Task and the *update step* is performed in the Reduce task. Centroid data is broadcasted at the beginning of each iteration. Intermediate data communication is relatively costly in KMeans clustering as each Map Task outputs data equivalent to the size of the centroids in each iteration.

Figure 4(a) presents the Twister4Azure KMeansClustering performance on different Azure compute instance types, with the number of map workers per instance equal to the number of cores of the instance. We did not notice any significant performance variations across the instances. Figure 4(b) shows that the performance scales well with the number of iterations. The performance improvement with a higher number of iterations in Figure 4(b) is due to the initial data download/parsing overhead distributing over the iterations. Figure 4(c) presents the number of map tasks executing at a given time throughout the job. The job consisted of 256 map tasks per iteration, generating 2 waves of map tasks per iteration. The dips represent the synchronization at the end of iterations. The gaps between the bars represent the total of overhead of the intermediate data communication, reduce task execution, merge task execution, data broadcasting and the new iteration scheduling that happens between iterations. According to the graph such overheads are relatively very

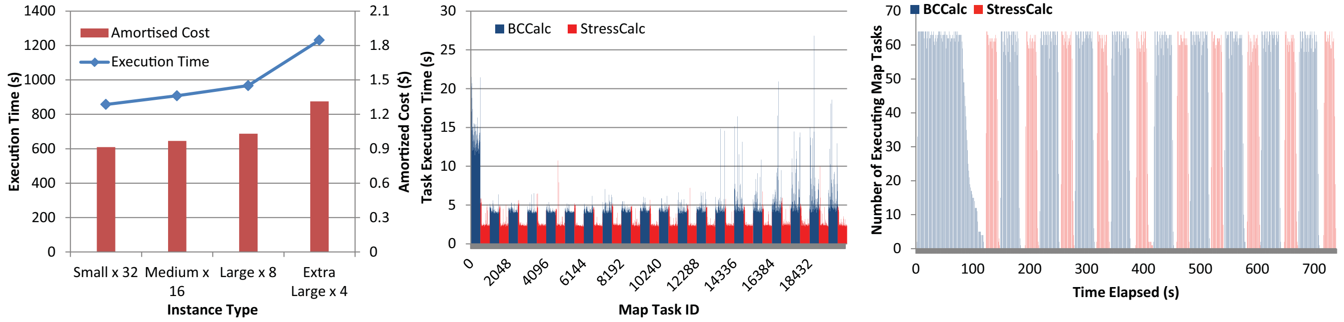

Figure 6. Twister4Azure MDS performance **Left:** Instance type study using 76800 data points, 32 instances, 20 iterations. **Center**: Twister4Azure MDS individual task execution time histogram for 144384 x144384 distance matrix in 64 Azure small instances, 10 iterations **Right:** Twister4Azure executing Map Task histogram for 144384 x144384 distance matrix in 64 Azure small instances, 10 iterations

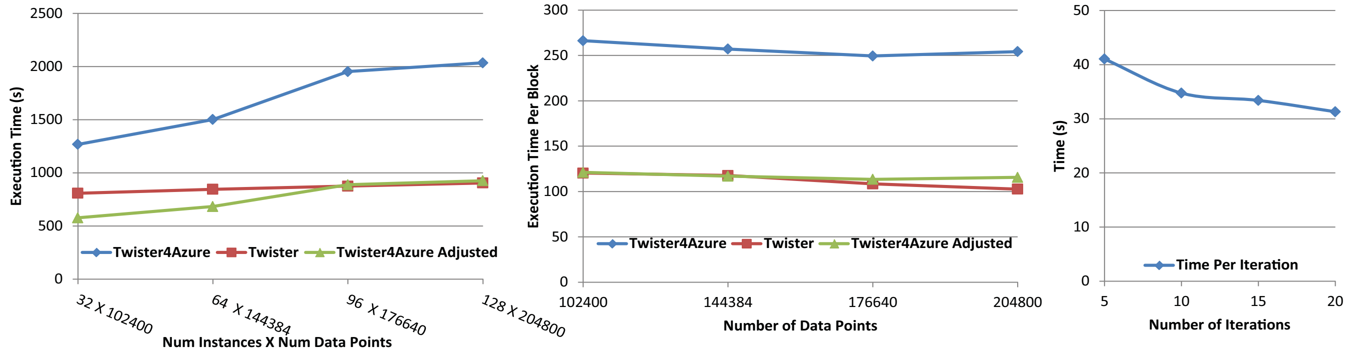

Figure 7. **Left:** Weak scaling where workload per core is ~constant. Ideal is a straight horizontal line. **Center :** Data size scaling with 128 Azure small instances/cores, 20 iterations. **Right**: Time per iteration with increasing number of iterations 30k x 30k distance matrix, 15 instances.

small. Figure 5(c) depicts the execution time of MapTasks across the whole job. The higher execution time of the tasks in the first iteration is due to the overhead of initial data downloading, parsing and loading, which is an indication of the performance improvement we get in subsequent iterations due to the data caching.

We also compared the Twister4Azure KMeansClustering performance with implementations of Java HPC Twister and Hadoop. The Java HPC Twister and Hadoop experiments were performed in a dedicated iDataPlex cluster of Intel(R) Xeon(R) CPU E5410 (2.33GHz) x 8 cores with 16GB memory per compute node with Gigabit Ethernet on Linux. Java HPCTwister results do not include the initial data distribution time. Figure 5(a) presents the relative (relative to the smallest parallel test in 32 instances) parallel efficiency of KMeansClustering for strong scaling, in which we keep the amount of data constant and increase the number of instances/cores. Figure 5(c) presents the execution time for weak scaling, wherein we increase the number of compute resources while keeping the work per core constant (work ~ number of nodes). We notice that Twister4Azure performance scales well up to 128 nodes in both experiments and shows minor performance degradation with 192 and 256 instances. The Twister4Azure adjusted (t_a) line in Figure 5(b) depicts the performance of Twister4Azure normalized according to the ratio between the Kmeans sequential performance in Azure (t_{sa}) and the Kmeans sequential performance in the cluster (t_{sc}) environment calculated using the t_a x (t_{sc}/t_{sa}) equation. This estimation, however, does not take into account the overheads which remain constant irrespective of the computation time. All tests we performed using 20 dimensional data and 500 centroids.

V. Multi Dimensional Scaling

The objective of multi-dimensional scaling (MDS) is to map a data set in high-dimensional space to a user-defined lower dimensional space with respect to the pairwise proximity of the data points[13]. Dimensional scaling is used mainly in the visualizing of high-dimensional data by mapping them to two or three dimensional space. MDS has been used to visualize data in diverse domains, including but not limited to bio-informatics, geology, information sciences, and marketing. We use MDS to visualize dissimilarity distances for hundreds of thousands of DNA and protein sequences to identify relationships.

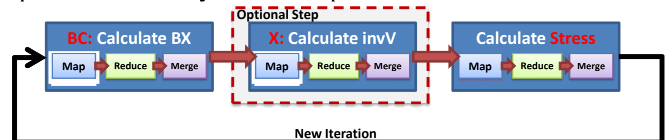

Figure 8. Twister4Azure Multi-Dimensional Scaling

In this paper we use Scaling by MAjorizing a COmplicated Function (SMACOF)[14], an iterative majorization algorithm. The input for MDS is an $N*N$ matrix of pairwise proximity values, where N is the number of data points in the high-dimensional space. The resultant lower dimensional mapping in D dimensions, called the X values, is an $N*D$ matrix.

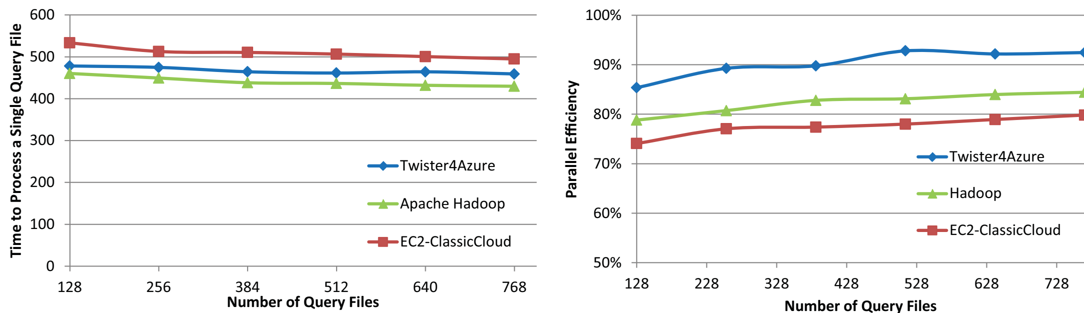

Figure 9. Twister4Azure BLAST performance. **Left** : Time to process a single query file. **Right**: Absolute parallel efficiency

The limits of MDS are more bounded by memory size than the CPU power. The main objective of parallelizing MDS is to leverage the distributed memory to support processing of larger data sets. In this paper, we implement the parallel SMACOF algorithm described by Bae et al[15]. This results in iterating a chain of 3 MapReduce jobs, as depicted in Figure 8. For the purposes of this paper, we perform an unweighted mapping that results in two MapReduce jobs steps per iteration, CalculateBC and CalculateStress. Each BCCalc Map task generates a portion of the total X matrix. MDS is more challenging for Twister4Azure due to its relatively finer grained task sizes and multiple MapReduce applications per iteration.

Figure 6(a) presents Twister4Azure MDS performance on different Azure compute instance types, with number of map workers per instance equal to number of cores of the instance. The performance degraded with the larger instances, which could be due to the memory bandwidth limitations. Figure 6(b) depicts the execution time of individual map-tasks for 10 iterations of MDS on 64 instances. The higher execution time of the tasks in the first iteration is due to the overhead of initial data downloading, parsing and loading. This overhead is relatively much higher in MDS (up to ~300% of task execution time vs ~60% in KMeans), enabling Twister4Azure to provide large performance gains relative to any non data-cached implementation. Figure 6(c) presents the number of map tasks executing at a given time for 10 iterations. The gaps between iterations are small, yet relatively larger than in KMeans which depicts that the between-iteration overheads are slightly larger for MDS. Also we can notice several tasks taking abnormally long execution times, slowing down the whole iteration. Figure 7(c) shows that the performance improves with a higher number of iterations due to the initial data download/parsing overhead getting distributed over the iterations.

We also compared the Twister4Azure MDS performance with Java HPC Twister MDS implementation. The Java HPC Twister experiment was performed in a dedicated large-memory cluster of Intel(R) Xeon(R) CPU E5620 (2.4GHz) x 8 cores with *192*GB memory per compute node with Gigabit Ethernet on *Linux*. Java HPCTwister results do not include the initial data distribution time. Figure 7(a) presents the execution time for weak scaling, where we increase the number of compute resources while keeping the work per core constant (work ~ number of cores). We notice that Twister4Azure exhibits acceptable encouraging performance. Figure 7(b) shows that MDS performance scales well with increasing data sizes. The *Twister4Azure adjusted* (t_a) line in Figure 7(a) and (b) depicts the performance of Twister4Azure normalized according to the sequential MDS BC calculation and Stress calculation performance ratio between the Azure(t_{sa}) and Cluster(t_{sc}) environments calculated using t_a x (t_{sc}/t_{sa}). This estimation however does not account for the overheads which remain constant irrespective of the computation time. In the above testing, the total number of tasks per job ranged from 10240 to 40960, proving Twister4Azure's ability to support large number of tasks effectively.

VI. SEQUENCE SEARCHING USING BLAST

NCBI BLAST+ [1] is the latest version of popular BLAST program, that is used to handle sequence similarity searching. Queries are processed independently and have no dependencies between them making it possible to use multiple BLAST instances to process queries in a pleasingly parallel manner. We performed the BLAST+ scaling speedup performance experiment from Gunarathne, et al[3] using Twister4Azure Blast+ to compare the performance with Amazon EC2 classic cloud and Apache Hadoop BLAST+ implementations. We used Azure Extra Large instances with 8 Map workers per node for the Twister4Azure BLAST experiments. We used a sub-set of a real-world protein sequence data set (100 queries per map task) as the input BLAST queries and used NCBI's non-redundant (NR) protein sequence database. All of the implementations downloaded and extracted the compressed BLAST database to a local disk of each worker prior to beginning processing of the tasks. Twister4Azure's ability to specify deploy time initialization routines was used to download and extract the database. The performance results do not include the database distribution times.

The Twister4Azure BLAST+ absolute efficiency (Figure 9) was better than the Hadoop and EMR implementations. Additionally the Twister4Azure performance was comparable to the performance of the Azure Classic Cloud BLAST results that we had obtained earlier. This shows that the performance of BLAST+ is sustained in Twister4Azure, even with the added complexity of MapReduce and iterative MapReduce.

VII. RELATED WORKS

CloudMapReduce[16] for Amazon Web Services (AWS) and Google AppEngine MapReduce[17] follow an architecture similar to MRRoles4Azure, in which they utilize the cloud services as the building blocks. Amazon

ElasticMapReduce[18] offers Apache Hadoop as a hosted service on the Amazon AWS cloud environment. However none of them support iterative MapReduce.

Haloop[19] extends Apache Hadoop to support iterative applications and supports caching of loop-invariant data as well as loop-aware scheduling. Spark[20] is a framework implemented using Scala to support interactive MapReduce like operations to query and process read-only data collections, while supporting in-memory caching and re-use of data products.

AzureBlast[21] is an implementation of parallel BLAST on Azure environment that uses Azure cloud services with an architecture similar to the Classic Cloud model, which is a predecessor to Twister4Azure. CloudClustering[22] is a prototype KMeansClustering implementation that uses Azure infrastructure services. CloudClustering uses multiple queues (single queue per worker) for job scheduling and supports caching of loop-invariant data.

VIII. Conclusion and Future Works

We have developed Twister4Azure, a novel iterative MapReduce distributed computing runtime for Azure cloud. We have implemented three important scientific applications using Twister4Azure – KmeansClustering, MDS and BLAST+. Twister4Azure enables the users to easily and efficiently perform large scale iterative data analysis for scientific applications on a commercial cloud platform.

In developing Twister4Azure, we encounter the challenges of scalability and fault tolerance unique to utilizing the cloud interfaces. We have developed a solution to support multi-level caching of loop-invariant data across iterations as well as caching of any reused data (e.g. broadcast data) and proposed a novel hybrid scheduling mechanism to perform cache-aware scheduling.

KmeansClustering and MDS are presented as iterative scientific applications of Twister4Azure. Experimental evaluation shows that Kmeans Clustering using Twster4Azure with virtual instances outperforms Apache Hadoop in local cluster by a factor of 2 to 4 and exhibits performance comparable to Java HPC Twister running on a local cluster. We consider the results presented in this paper as one of the first or the first large-scale study of Azure performance for non-trivial scientific applications.

Twister4Azure and Java HPC Twister illustrate our roadmap to a cross platform new programming paradigm supporting large scale data analysis, an important area for both HPC and eScience applications.

Acknowledgment

This work is funded in part by the Microsoft Azure Grant. We would like to thank Prof. Geoffrey C Fox for his many insights and feedbacks about this work. We would also like to thank Seung-Hee Bae for many discussions on MDS.

References

[1] G. C. C. Camacho, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer and T. L. Madden, "BLAST+: architecture and applications," *BMC Bioinformatics 2009, 10:421,* 2009.

[2] J. Ekanayake, T. Gunarathne, and J. Qiu, "Cloud Technologies for Bioinformatics Applications," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 22, pp. 998-1011, 2011.

[3] T. Gunarathne, T.-L. Wu, J. Y. Choi, S.-H. Bae, and J. Qiu, "Cloud computing paradigms for pleasingly parallel biomedical applications," *Concurrency and Computation: Practice and Experience,*, 2011. doi: 10.1002/cpe.1780

[4] T. Gunarathne, W. Tak-Lon, J. Qiu, and G. Fox, "MapReduce in the Clouds for Science," in *IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 565-572.

[5] "*Twister4Azure"*. http://salsahpc.indiana.edu/twister4azure/(7/25/11)

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM,* vol. 51, pp. 107-113, 2008.

[7] "Apache Hadoop,": http://hadoop.apache.org/core/. (7/25/11)

[8] J.Ekanayake, H.Li, B.Zhang, T.Gunarathne, S.Bae, J.Qiu, and G.Fox, "Twister: A Runtime for iterative MapReduce," 1st Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010, Chicago, Illinois.

[9] B. Zhang, Y. Ruan, T.L. Wu, J. Qiu, A. Hughes, and G.C. Fox, "Applying Twister to Scientific Applications," presented at the CloudCom 2010, IUPUI Conference Center Indianapolis, 2010.

[10] "Apache ActiveMQ" : http://activemq.apache.org/(7/25/2011)

[11] "*Windows Azure Compute*." (7/25/2011) http://www.microsoft.com/windowsazure/features/compute/

[12] S. Lloyd, "Least squares quantization in PCM," *Information Theory, IEEE Transactions on,* vol. 28, pp. 129-137, 1982.

[13] J. B. Kruskal and M. Wish, *Multidimensional Scaling*: Sage Publications Inc., 1978.

[14] J. Leeuw, "Convergence of the majorization method for multidimensional scaling," *Journal of Classification,* vol. 5, pp.163, 1988.

[15] S.H. Bae, J.Y. Choi, J. Qiu, and G. C. Fox, "Dimension reduction and visualization of large high-dimensional data via interpolation," 19th ACM International Symposium on High Performance Distributed Computing, Chicago, Illinois, 2010.

[16] "cloudmapreduce,"http://code.google.com/p/cloudmapreduce/ (8/20/10)

[17] "*AppEngine MapReduce"*.: http://code.google.com/p/appengine-mapreduce (8/20/2010)

[18] "Amazon Web Services,": http://aws.amazon.com/. (7/25/2011)

[19] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient Iterative Data Processing on Large Clusters,", 36th International Conference on Very Large Data Bases, Singapore, 2010.

[20] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," presented at the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10), Boston, 2010.

[21] W. Lu, J. Jackson, and R. Barga, "AzureBlast: A Case Study of Developing Science Applications on the Cloud," 1st Workshop on Scientific Cloud Computing , HPDC.Chicago, IL, 2010.

[22] A. Dave, W. Lu, J. Jackson, and R. Barga, "CloudClustering: Toward an iterative data processing pattern on the cloud," in *First International Workshop on Data Intensive Computing in the Clouds*, Anchorage, Alaska, 2011.