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.

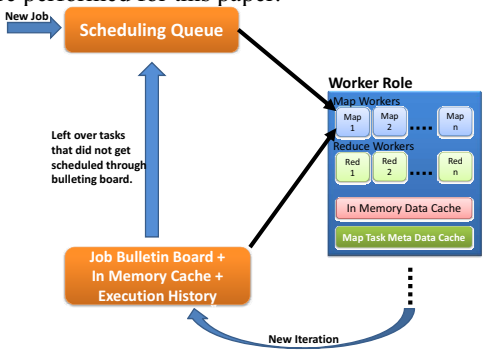

Figure 3. Cache Aware Hybrid Scheduling

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

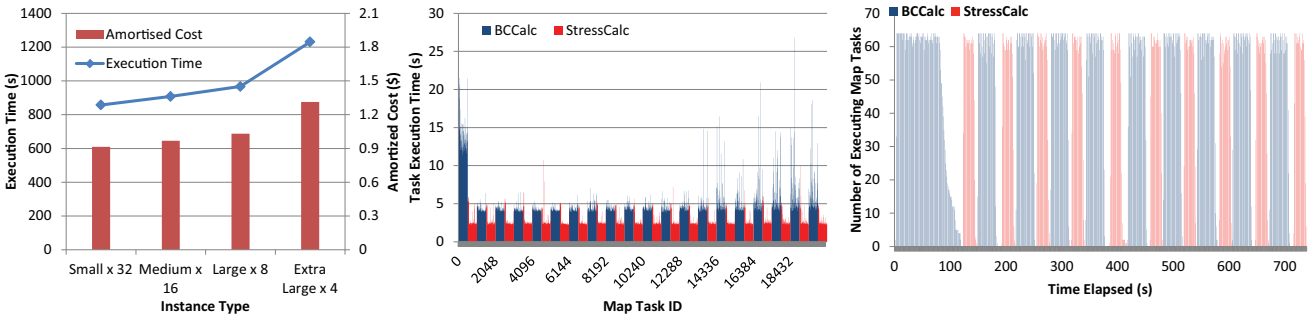

Figure 6. Twister4Azure MDS performance **Left:** Instance type study using 76800 data points, 32 instances, 20 iterations. **Center**: Twister4Azure MDS individual task execution time histogram for 144384 x144384 distance matrix in 64 Azure small instances, 10 iterations **Right:** Twister4Azure executing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

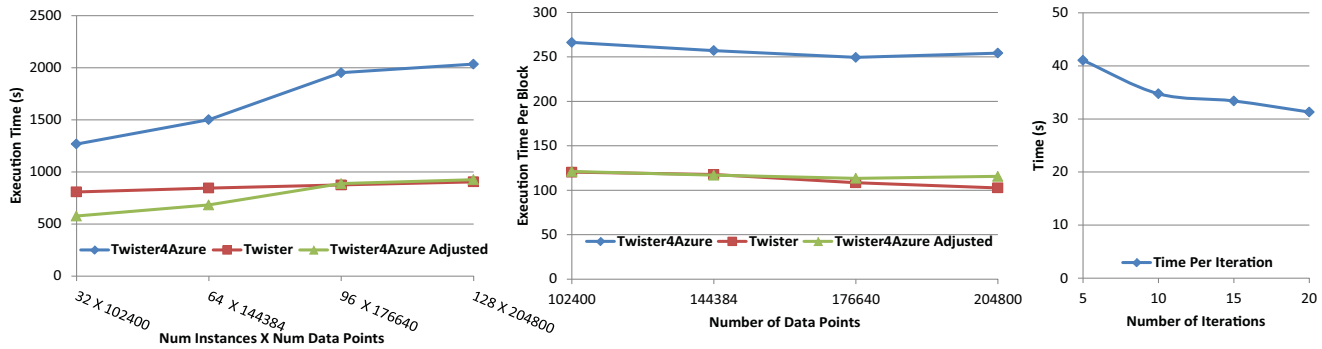

Figure 7. **Left:** Weak scaling where workload per core is ~constant. Ideal is a straight horizontal line. **Center :** Data size scaling with 128 Azure small instances/cores, 20 iterations. **Right**: Time per iteration with increasing number of iterations 30k x 30k distance matrix, 15 instances.

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.

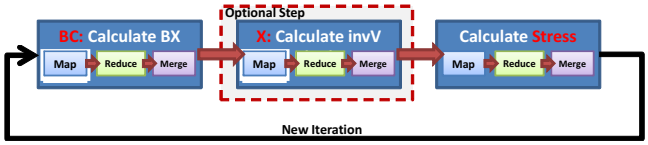

Figure 8. Twister4Azure Multi-Dimensional Scaling

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.

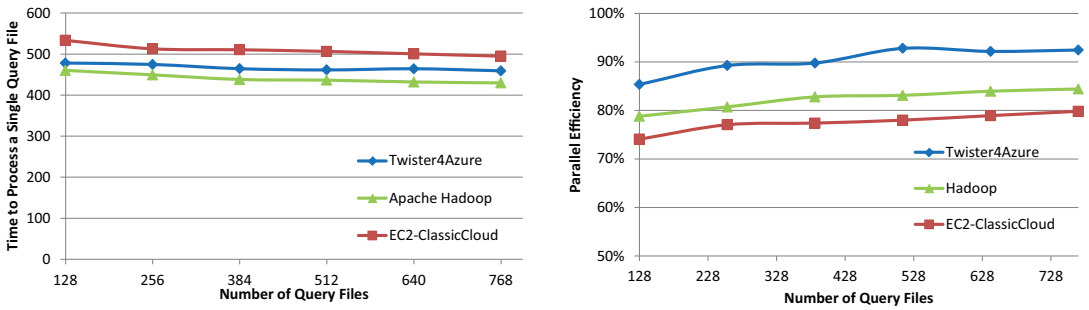

Figure 9. Twister4Azure BLAST performance. **Left** : Time to process a single query file. **Right**: Absolute parallel efficiency

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.