# MapReduce in the Clouds for Science

Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, Geoffrey Fox
School of Informatics and Computing / Pervasive Technology Institute
Indiana University, Bloomington.
{tgunarat, taklwu, xqiu,gcf}@indiana.edu

*Abstract*— **The utility computing model introduced by cloud computing combined with the rich set of cloud infrastructure services offers a very viable alternative to traditional servers and computing clusters. MapReduce distributed data processing architecture has become the weapon of choice for data-intensive analyses in the clouds and in commodity clusters due to its excellent fault tolerance features, scalability and the ease of use. Currently, there are several options for using MapReduce in cloud environments, such as using MapReduce as a service, setting up one's own MapReduce cluster on cloud instances, or using specialized cloud MapReduce runtimes that take advantage of cloud infrastructure services. In this paper, we introduce AzureMapReduce, a novel MapReduce runtime built using the Microsoft Azure cloud infrastructure services. AzureMapReduce architecture successfully leverages the high latency, eventually consistent, yet highly scalable Azure infrastructure services to provide an efficient, on demand alternative to traditional MapReduce clusters. Further we evaluate the use and performance of MapReduce frameworks, including AzureMapReduce, in cloud environments for scientific applications using sequence assembly and sequence alignment as use cases.**

***Keywords- MapReduce, Cloud Computing, AzureMapReduce, Elastic MapReduce, Hadoop***

## I. INTRODUCTION

Today, scientific research is increasingly reliant on the processing of very large amounts of data, which Jim Gray has dubbed the 4th paradigm [1]. Scientists are more reliant on computing resources than ever and desire more power and greater ease of use. At the same time, we notice that the industry's introduction of the concepts of Cloud Computing and MapReduce gives scientists very viable alternatives for their computational needs. In fact, the utility computing model offered by cloud computing is remarkably well-suited for scientists' staccato computing needs [2]. While clouds offer raw computing power combined with cloud infrastructure services offering storage and other services, there is a need for distributed computing frameworks to harness the power of clouds both easily and effectively. At the same time, it should be noted that cloud infrastructures are known to be less reliable than their traditional cluster counterparts and do not provide the high-speed interconnects needed by frameworks such as MPI.

The MapReduce distributed data analysis framework model introduced by Google [3] provides an easy-to-use programming model that features fault tolerance, automatic parallelization, scalability and data locality-based optimizations. Due to their excellent fault tolerance features, MapReduce frameworks are well-suited for the execution of large distributed jobs in brittle environments such as commodity clusters and cloud infrastructures. Though introduced by the industry and used mainly in the information retrieval community, it is shown [4-6] that MapReduce frameworks are capable of supporting many scientific application use cases, making these frameworks good choices for scientists to easily build large, data-intensive applications that need to be executed within cloud infrastructures.

The Microsoft Azure platform currently does not provide or support any distributed parallel computing frameworks such as MapReduce, Dryad or MPI, other than the support for implementing basic queue-based job scheduling. Also, the platform-as-a-service nature of Azure makes it difficult or rather impossible to set up an existing general purpose runtime on Windows Azure instances. This lack of a distributed computing framework on Azure platform motivated us to implement AzureMapReduce, which is a decentralized novel MapReduce run time built using Azure cloud infrastructure services. AzureMapReduce implementation takes advantage of the scalability, high availability and the distributed nature of cloud infrastructure services, guaranteed by cloud service provider, to deliver a fault tolerant, dynamically scalable runtime with a familiar programming model for the users.

Several options exist for executing MapReduce jobs on cloud environments, such as manually setting up a MapReduce (e.g.: Hadoop [7]) cluster on a leased set of computing instances, using an on-demand MapReduce-as-service offering such as Amazon ElasticMapReduce (EMR) [8] or using a cloud MapReduce runtime such as AzureMapReduce or CloudMapReduce [9]. In this paper we explore and evaluate each of these different options for two well-known bioinformatics applications: Smith-Waterman GOTOH pairwise distance alignment (SWG) [10-11], Cap3 [12] sequence assembly. We have performed experiments to gain insight about the performance of MapReduce in the clouds for the selected applications and compare its performance to MapReduce on traditional clusters. For this study, we use an experimental version of AzureMapReduce.

Our work was motivated by an experience we had in early 2010 in which we evaluated the use of Amazon EMR for our scientific applications. To our surprise, we observed subpar performance in EMR compared to using a manually-built cluster on EC2 (which is not the case anymore), which prompted us to perform the current analyses. In this paper, we show that MapReduce computations performed in cloud

environments, including AzureMapReduce, has the ability to perform comparably to MapReduce computations on dedicated private clusters.

## II. TECHNOLOGIES

### A. Hadoop

Apache Hadoop [7] MapReduce is an open-source MapReduce style distributed data processing framework, which is an implementation similar to the Google MapReduce [3]. Apache Hadoop MapReduce uses the HDFS [13] distributed parallel file system for data storage, which stores the data across the local disks of the computing nodes while presenting a single file system view through the HDFS API. HDFS is targeted for deployment on unreliable commodity clusters and achieves reliability through the replication of file data. When executing Map Reduce programs, Hadoop optimizes data communication by scheduling computations near the data by using the data locality information provided by the HDFS file system. Hadoop has an architecture consisting of a master node with many client workers and uses a global queue for task scheduling, thus achieving natural load balancing among the tasks. The Map Reduce model reduces the data transfer overheads by overlapping data communication with computations when reduce steps are involved. Hadoop performs duplicate executions of slower tasks and handles failures by rerunning the failed tasks using different workers.

### B. Amazon Web Services

Amazon Web Services (AWS) [14] is a set of on-demand, over the wire cloud computing services offered by Amazon. AWS offers a wide range of computing, storage and communication services including, but not limited to, Elastic Compute Cloud (EC2), Elastic MapReduce (EMR), Simple Storage Service (S3) and Simple Queue Service (SQS).

The Amazon EC2 service enables users to lease Xen based virtual machine instances over the internet, billed hourly with the use of a credit card, allowing users to dynamically provision resizable virtual clusters in a matter of minutes. EC2 is offered as an infrastructure as a service approach, wherein the users get direct access to the virtual machine instances. EC2 provides users with the capability to store virtual machine snapshots in the form of Amazon Machine Images (AMI), which can be used to launch instances at a later time. EC2 offers a rich variety of instance types based on the computing capacity, memory and the I/O performance, each with a different price point, allowing users to be economical and flexible by choosing the best matching instances for their use case. In [15] we perform an analysis of a select set of EC2 instance types. EC2 offers both Linux instances as well as Windows instances.

### C. Amazon Elastic Map Reduce

Amazon Elastic MapReduce (EMR) [14] provides MapReduce as an on-demand service hosted within the Amazon infrastructure. EMR is a hosted Apache Hadoop [7] MapReduce framework, which utilizes Amazon EC2 for computing power and Amazon S3 for data storage. It allows the users to perform Hadoop MapReduce computations in the cloud with the use of a web application interface, as well as a command line API, without worrying about installing and configuring a Hadoop cluster. Users can run their existing Hadoop MapReduce program on EMR with minimal changes.

EMR supports the concept of JobFlows, which can be used to support multiple steps of Map & Reduce on a particular data set. Users can specify the number and the type of instances that are required for their Hadoop cluster. For EMR clusters consisting of more than one instance, EMR uses one instance exclusively as the Hadoop master node. EMR does not provide the ability to use custom AMI images. As an alternative to custom AMI images, EMR provides Bootstrap actions that users can specify to run on the computing nodes prior to the launch of the MapReduce job, which can be used for optional custom preparation of the images. EMR allows for debugging of EMR jobs by providing the option to upload the Hadoop log files in to S3 and state information to SimpleDB. Intermediate data and temporary data are stored in the local HDFS file system while the job is executing. Users can use either the S3 native (s3n) file system or the legacy S3 block file system to specify input and output locations on Amazon S3. Use of s3n is recommended, as it allows files to be saved in native formats in S3.

The pricing for the use of EMR consists of the cost for the EC2 computing instances, the S3 storage cost, an optional fee for the usage of SimpleDB to store job debugging information, and a separate cost per instance hour for the EMR service.

### D. Microsoft Azure Platform

The Microsoft Azure platform [16] is a cloud computing platform that offers a set of cloud computing services similar to the Amazon Web Services. Windows Azure Compute allows the users to lease Windows virtual machine instances. Azure Compute follows a platform as a service approach and offers the .net runtime as the platform. Users can deploy their programs as an Azure deployment package through a web application. Unlike Amazon EC2, in Azure users do not have the ability to interact directly with the Azure instances, other than through the deployed programs. But on the other hand, platform-as-a-service infrastructures have a greater capability to offer quality of service and automated management services than infrastructure-as-a-service offerings. Azure offers a limited set of instances on a linear price and feature scale.

The Azure storage queue is an eventual consistent, reliable, scalable and distributed web-scale message queue service, ideal for small, short-lived, transient messages. This messaging framework can be used as a message-passing mechanism to communicate between distributed components of any application running in the cloud. Messages can only contain text data and the size is limited to 8KB per message. Users can create an unlimited number of queues and send an unlimited number of 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 the 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. The message will reappear upon expiration of the timeout, as long as the previous reader did not delete it.

The Azure Storage BLOB service provides a web-scale distributed storage service where users can store and retrieve any type of data through a web services interface. Azure Blob service offers two types of blobs, namely block blobs, which are optimized for streaming access, and page blobs, which are optimized for random access.

## III. Challenges for MapReduce in the Clouds

As mentioned in the introduction, MapReduce frameworks perform much better in brittle environments than other tightly coupled distributed programming frameworks, such as MPI [17], due to their excellent fault tolerance capabilities. However, cloud environments provide several challenges for MapReduce frameworks to harness the best performance.

- Data storage: Clouds typically provide a variety of storage options, such as off-instance cloud storage (e.g.: Amazon S3), mountable off-instance block storage (e.g.: Amazon EBS) as well as virtualized instance storage (persistent for the lifetime of the instance), which can be used to set up a file system similar to HDFS [13]. The choice of the storage best-suited to the particular MapReduce deployment plays a crucial role as the performance of data intensive applications rely a lot on the storage location and on the storage bandwidth.
- Metadata storage: MapReduce frameworks need to maintain metadata information to manage the jobs as well as the infrastructure. This metadata needs to be stored reliability ensuring good scalability and the accessibility to avoid single point of failures and performance bottlenecks to the MapReduce computation.
- Communication consistency and scalability: Cloud infrastructures are known to exhibit inter-node I/O performance fluctuations (due to shared network, unknown topology), which affect the intermediate data transfer performance of MapReduce applications.
- Performance consistency (sustained performance): Clouds are implemented as shared infrastructures operating using virtual machines. It's possible for the performance to fluctuate based the load of the underlying infrastructure services as well as based on the load from other users on the shared physical node which hosts the virtual machine (see Section VII).
- Reliability (Node failures): Node failures are to be expected whenever large numbers of nodes are utilized for computations. But they become more prevalent when virtual instances are running on top of non-dedicated hardware. While MapReduce frameworks can recover jobs from worker node failures, master node (nodes which store meta-data, which handle job scheduling queue, etc) failures can become disastrous.
- Choosing a suitable instance type: Clouds offer users several types of instance options, with different configurations and price points (See Sections B and D). It's important to select the best matching instance type, both in terms of performance as well as monetary wise, for a particular MapReduce job.
- Logging: Cloud instance storage is preserved only for the lifetime of the instance. Hence, information logged to the instance storage would be lost after the instance termination. This can be crucial if one needs to process the logs afterwards, for an example to identify a software-caused instance failure. On the other hand, performing excessive logging to a bandwidth limited off-instance storage location can become a performance bottleneck for the MapReduce computation.

## IV. AzureMapReduce

AzureMapReduce is a distributed decentralized MapReduce runtime for Windows Azure that was developed using Azure cloud infrastructure services. The usage of the cloud infrastructure services allows the AzureMapReduce implementation to take advantage of the scalability, high availability and the distributed nature of such services guaranteed by the cloud service providers to avoid single point of failures, bandwidth bottlenecks (network as well as storage bottlenecks) and management overheads. In this paper, we use an experimental, pre-release version of AzureMapReduce.

The usage of cloud services usually introduces latencies larger than their optimized non-cloud counterparts and often does not guarantee the time for the data's first availability. These overheads can be conquered, however, by using a sufficiently coarser grained map and reduce tasks. AzureMapReduce overcomes the availability issues by retrying and by designing the system so it does not rely on the immediate availability of data to all the workers. In this paper, we use the pre-release implementation of the AzureMapReduce runtime, which uses Azure Queues for map and reduce task scheduling, Azure tables for metadata & monitoring data storage, Azure blob storage for input, output and intermediate data storage and the Window Azure Compute worker roles to perform the computations.

Google MapReduce [3], Hadoop [7] as well as Twister [18] MapReduce computations are centrally controlled using a master node and assume master node failures to be rare. In those run times, the master node handles the task assignment, fault tolerance and monitoring for the completion of Map and Reduce tasks, in addition to other responsibilities. By design, cloud environments are more brittle than the traditional computing clusters are. Thus, cloud applications should be developed to anticipate and withstand these failures. Because of this, it is not possible for AzureMapReduce to make the same assumptions of reliability about a master node as in the above-mentioned runtimes. Due to these reasons, AzureMapReduce is designed around a decentralized control model without a master node, thus avoiding the possible single point of failure. AzureMapReduce also provides users with the capability to dynamically scale up or down the number of

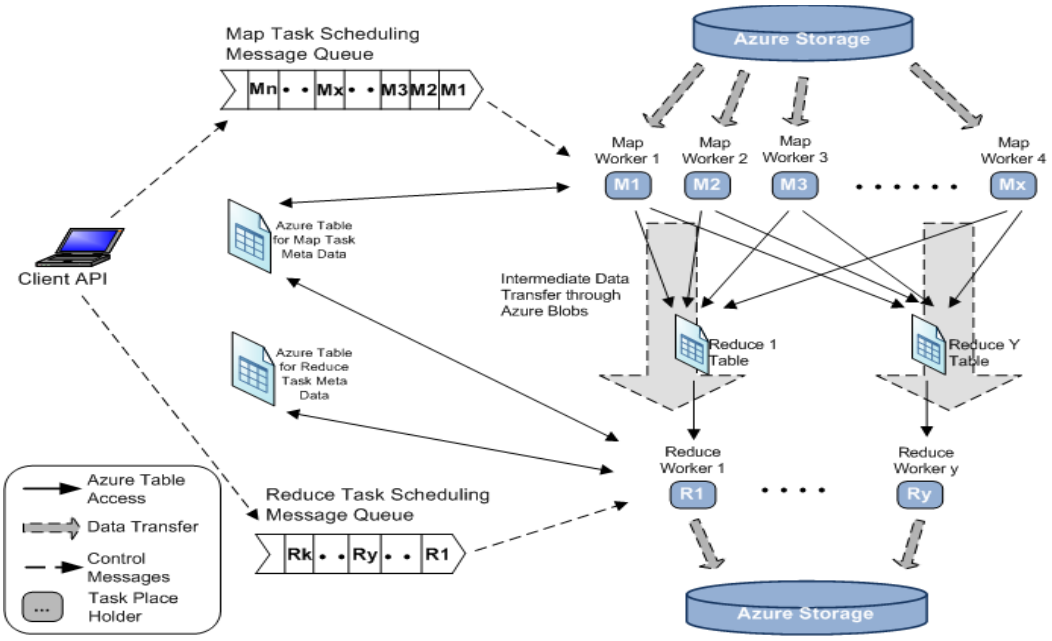


Figure 1. AzureMapReduce Architecture

computing instances, even in the middle of a MapReduce computation, as and when it is needed. The map and reduce tasks of the AzureMapReduce runtime are dynamically scheduled using a global queue. In a previous study [19], we experimentally showed that dynamic scheduling through a global queue achieves better load balancing across all tasks, resulting in better performance and throughput than statically scheduled runtimes, especially when used with real-world inhomogeneous data distributions.

### A. *Client API and Driver*

Client driver is used to submit the Map and Reduce tasks to the worker nodes using Azure Queues. Users can utilize the client API to generate a set of map tasks that are either automatically based on a data set present in the Azure Blob storage or manually based on custom criteria, which we find to be a very useful feature when implementing science applications using MapReduce. Client driver uses the .net task parallel library to dispatch tasks in parallel overcoming the latencies of the Azure queue and the Azure table services. It's possible to use the client driver to monitor the progress and completion of the MapReduce jobs.

### B. *Map Tasks*

Users have the ability to configure the number of Map workers per Azure instance. Map workers running on the Azure compute instances poll and dequeue map task scheduling messages from the scheduling queue, which were enqueued by the client API. The scheduling messages contain the meta-data needed for the Map task execution, such as input data file location, program parameters, map task ID, and so forth. Map tasks upload the generated intermediate data to the Azure Blob Storage and put the key-value pair meta-data information to the correct reduce task table. At this time, we are actively working on investigating other approaches for performing the intermediate data transfer.

### C. *Reduce Tasks*

Reduce task scheduling is similar to map task scheduling. Users have the ability to configure the number of Reduce tasks per Azure Compute instance. Each reduce task has an associated Azure Table containing the input key-value pair meta-data information generated by the map tasks. Reduce tasks fetch intermediate data from the Azure Blob storage based on the information present in the above-mentioned reduce task table. This data transfer begins as soon as the first Map task is completed, overlapping the data transfer with the computation. This overlapping of data transfer with computation minimizes the data transfer overhead of the MapReduce computations, as found in our testing. Each Reduce task starts processing the reduce phase; when all the map tasks are completed, and after all the intermediate data products bound for that particular reduce task is fetched. In the AzureMapReduce, each reduce task will independently determine the completion of map tasks based on the information in the map task meta-data table and in the reduce task meta-data table. After completion of the processing, reduce tasks upload the results to the Azure Blob Storage and update status in the reduce task meta-data table.

Azure table does not support transactions across tables or guarantee the immediate availability of data, but rather guarantees the eventual availability data. Due to that, it is possible for a worker to notice a map task completion update, before seeing a reduce task intermediate meta-data record added by that particular map task. Even though rare, this can result in an inconsistent state where a reduce task decides all the map tasks have been completed and all the intermediate data bound for that task have been transferred successfully, while in reality it's missing some intermediate

data items. In order to remedy this, map tasks store the number of intermediate data products it generated in the map task meta-data table while doing the task completion status update. Before proceeding with the execution, reduce tasks perform a global count of intermediate data products in all reduce task tables and tally it with the total of intermediate data products generated by the map tasks. This process ensures all the intermediate data products are transferred before starting the reduce task processing.

### D. Monitoring

We use Azure tables for the monitoring of the map and reduce task meta-data and status information. Each job has two separate Azure tables for map and reduce tasks. Both the meta-data tables are used by the reduce tasks to determine the completion of Map task phase. Other than the above two tables, it is possible to monitor the intermediate data transfer progress using the tables for each reduce task.

### E. Fault Tolerance

Fault tolerance is achieved using the fault tolerance features of the Azure queue. When fetching a message from an Azure queue, a visibility timeout can be specified, which will keep the message hidden until the timeout expires. In Azure Map Reduce, map and reduce tasks delete messages from the queue only after successful completion of the tasks. If a task fails or is too slow processing, then the message will reappear in the queue after the timeout. In this case, it would be fetched and re-executed by a different worker. This is made possible by the side effect-free nature of the MapReduce computations as well as the fact that AzureMapReduce stores each generated data product in persistent storage, which allows it to ignore the data communication failures. In the current implementation, we retry each task three times before declaring the job a failure. We use the Map & Reduce task meta-data tables to coordinate the task status and completion. Over the course of our testing, we were able to witness few instances of jobs being recovered by the fault tolerance.

### F. Limitations of Azure MapReduce

Currently Azure allows a maximum of 2 hours for queue message timeout, which is not enough for Reduce tasks of larger MapReduce jobs, as the Reduce tasks typically execute from the beginning of the job till the end of the job. In our current experiments, we disabled the reduce tasks fault tolerance when it is probable for MapReduce job to execute for more than 2 hours. Also in contrast to Amazon Simple Queue Service, Azure Queue service currently doesn't allow for dynamic changes of visibility timeouts, which would allow for richer fault tolerance patterns.

### G. Related technologies

*1) Google AppEngine-MapReduce:* Google AppEngine-MapReduce [20] is an open source library aimed at performing MapReduce computations on the Google AppEngine platform using AppEngine services. The current experimental release only contains a Mapper library, while supporting the Reduce phase is part of planned enhancements. AppEngine-MapReduce supports Hadoop API with few minimal changes, allowing users to easily port the existing Hadoop applications to AppEngine-MapReduce applications. Users will not be able spawn proceses, restricting the use of executables, or threads.

*2) Cloud map reduce:* CloudMapReduce [9] also implements a decentralized MapReduce architecture using the cloud infrastructure services of Amazon Web Services. The main differences between CloudMapReduce and the pre-release version of AzureMapReduce lies in the method of handling the intermediate data and meta-data, and in the timing of the commencement of the reduce tasks, among others.

## V. Performance evaluation of MapReduce in the clouds for science

### A. Methodology

We performed scalability tests using the selected applications to evaluate the performance of the MapReduce implementations in the cloud environments, as well as in the local clusters. For the scalability test, we decided to increase the workload and the number of nodes proportionally (weak scaling), so that the workload per node remained constant.

All of the AzureMapReduce tests were performed using Azure small instances (one CPU core). The Hadoop-Bare Metal tests were performed on an iDataplex cluster, in which each node had two 4-core CPUs (Intel Xeon CPU E5410 2.33GHz) and 16 GB memory, and was inter-connected using Gigabit Ethernet network interface. The Hadoop-EC2 and EMR tests for Cap3 application were performed using Amazon High CPU extra-large instances, as they are the most economical per CPU core. Each high CPU extra-large instance was considered as 8 physical cores, even though they are billed as 20 Amazon compute units. The EC2 and EMR tests for SWG MapReduce applications were performed using Amazon extra-large instances as the more economical high CPU extra instances showed memory limitations for the SWG calculations. Each extra-large instance was considered as 4 physical cores, even though they are billed as 8 Amazon computing units. In all the Hadoop-based experiments (EC2, EMR and Hadoop bare metal), only the cores of the Hadoop slave nodes were considered for the number of cores calculation, despite the fact that an extra computing node was used as the Hadoop master node.

Below are the defined parallel efficiency and relative parallel efficiency calculations used in the results part of this paper.

$$Parallel\ Efficiency\ (Ep) = \frac{T(1)}{pT(\rho)}$$

T(1) is the best sequential execution time for the application in a particular environment using the same data set or a representative subset. In all the cases, the sequential time was measured with no data transfers, i.e. the input files were present in the local disks. T(ρ) is the parallel run time for the application while "p" is the number of processor cores used.

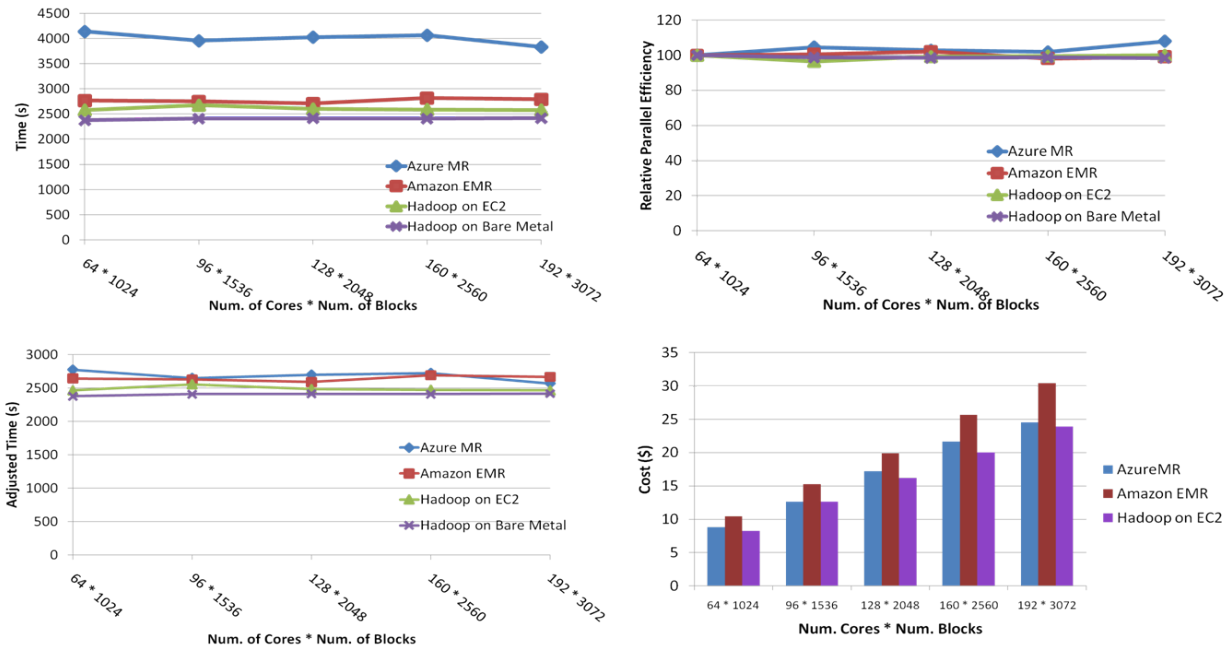


Figure 2. (a) SWG MapReduce pure performance (b) SWG MapReduce relative parallel efficiency (c) SWG MapReduce normalized performance (d) SWG MapReduce amortized cost for clouds

We calculate that the relative parallel efficiency when estimating the sequential run time for an application is not straightforward. $\alpha = p/p1$, where p1 is the smallest number of CPU cores for the experiment.

$$Relative\ Parallel\ Efficiency\ (Ep) = \frac{T(p1)}{\alpha T(p)}$$

### B. *Smith-Waterman-GOTOH(SWG) pairwise distance calculation*

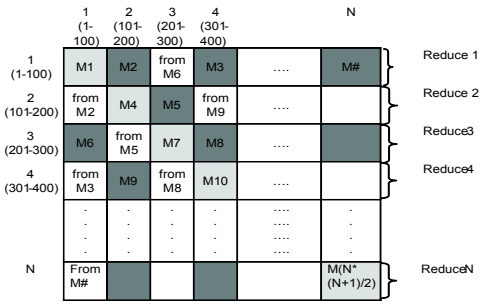


Figure 3. SWG MapReduce task decomposition

In this application, we use the Smith-Waterman [10] algorithm with GOTOH [11] (SWG) improvement to perform pairwise sequence alignment on FASTA sequences. Given a sequence set we calculate the all-pairs dissimilarity for all the sequences. When calculating the all-pairs dissimilarity for a data set, calculating only the strictly upper or lower triangular matrix in the solution space is sufficient, as the transpose of the computed triangular matrix gives the dissimilarity values for the other triangular matrix. As shown in figure 3, this property, together with blocked decomposition, is used when calculating the set of map tasks for a given job. Reduce tasks aggregate the output from a row block. In this application, the size of the input data set is relatively small, while the size of the intermediate and the output data are significantly larger due to the $n^2$ result space, stressing the performance of inter-node communication and output data storage. SWG can be considered as a memory-intensive application.

More details about the Hadoop-SWG application implementation are given in [19]. The AzureMapReduce implementation also follows the same architecture and blocking strategy as in the Hadoop-SWG implementation. Hadoop-SWG uses the open source JAligner [21] as the computational kernel, while AzureMapReduce SWG uses the C# implementation, NAligner [21] as the computational kernel. The results of the SWG MapReduce computation get stored in HDFS for Hadoop-SWG in bare metal and EC2 environments, while the results get stored in Amazon S3 and Azure Block Storage for Hadoop-SWG on EMR and SWG on AzureMapReduce, respectively.

Due to the all-pairs nature and the block-based task decomposition of the SWG MapReduce implementations, it's hard to increase the workload linearly by simply replicating the number of input sequences for the scalability test. Hence, we modified the program to artificially reuse the computational blocks of the smallest test case in the larger test cases, so that the workload scaling occurs linearly. The raw performance results of the SWG MapReduce scalability test are given in figure 2(a). A block size of 200 * 200 sequences is used in the performance experiments resulting in 40,000 sequence alignments per block, which resulted in ~123 million sequence comparisons in the 3072 block test case. The AzureMapReduce SWG performance in figure 2 (a) is significantly lesser than the others. This is due to the performance of NAligner core executing in windows

environment being slower than the JAligner core executing in Linux environment.

Due to the sheer size of even the smallest computation in our SWG scaling test cases, we found it impossible to calculate the sequential execution time for the SWG test cases. Also, due to the all-pairs nature of SWG, it's not possible to calculate the sequential execution time using a subset of data. In order to compensate for the lack of absolute efficiency (which would have negated most of the platform and hardware differences across different environments), we performed a moderately-sized sequential SWG calculation in all of the environments and used that result to normalize the performance using the Hadoop-bare metal performance as the baseline. The normalized performance is depicted in figure 2(c), where we can observe that all four environments show comparable performance and good scalability for the SWG application. Figure 2(b) depicts the relative parallel efficiency of SWG MapReduce implementations using the 64 core, 1024 block test case as *p1* (see section V-A).

In figure 2(d) we present the approximate computational costs for the experiments performed using cloud infrastructures. Even though the cloud instances are hourly billed, costs presented in 2(d) are amortized for the actual execution time (time / 3600 * num_instances * instance price per hour), assuming the remaining time of the instance hour has been put to useful work. In addition to the depicted charges, there will be additional minor charges for the data storage for EMR & AzureMapReduce. There will also be additional minor charges for the queue service and table service for AzureMapReduce. We notice that the costs for Hadoop on EC2 and AzureMapReduce are in a similar range, while EMR costs a fraction more. We consider the ability to perform a large computation, such as ~123 million sequence alignments, for under 30$ with zero up front hardware cost, as a great enabler for the scientists, who don't have access to in house compute clusters.

### C. *Sequence assembly using Cap3*

Cap3 [12] is a sequence assembly program which assembles DNA sequences by aligning and merging sequence fragments to construct whole genome sequences. The Cap3 algorithm operates on a collection of gene sequence fragments, which are presented as FASTA-formatted files, generating consensus sequences. The Cap3 program is often used in parallel with lots of input files due to the pleasingly parallel nature of the application. The size of a typical data input file for Cap3 program and the resulting data file range from hundreds of kilobytes to a few megabytes. The output files can be collected independently and do not need any combining steps. We use a Mapper-only MapReduce application for Cap3. More details about the Cap3 Hadoop implementation can be found on [15]. Cap3 can be considered as a CPU intensive application.

We used a replicated set of Fasta files as the input data in our experiments. Every file contained 458 reads. The input/output data was stored in HDFS in the Hadoop BareMetal and Hadoop-EC2 experiments, while they were stored in Amazon S3 and Azure Blob storage for EMR and

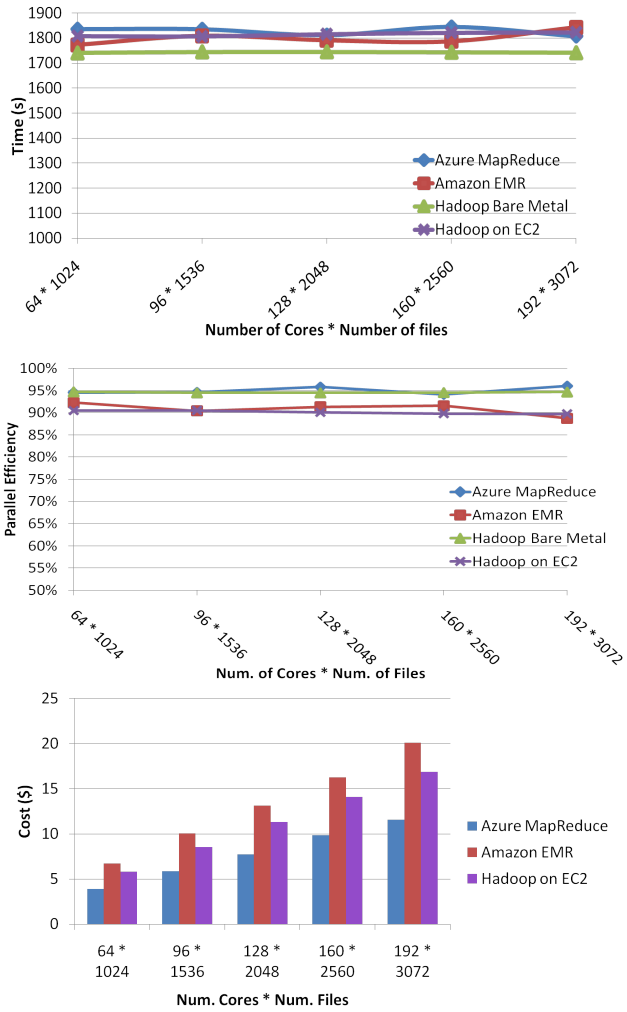


Figure 4. (a) Cap3 MapReduce scaling performance
(b) Cap3 MapReduce parallel efficiency
(c) Cap3 MapReduce computational cost in cloud infrastructures

AzureMapReduce experiments respectively. Figure 4(a) presents the pure performance of the Cap3 MapReduce applications, while Figure 4(b) presents the absolute parallel efficiency for the Cap3 MapReduce applications. As we can see, all of the cloud Cap3 applications displayed performance comparative to the bare metal clusters and good scalability, while AzureMapReduce and Hadoop Bare metal showed a slight edge over the Amazon counterparts in terms of the efficiency. Figure 4(c) depicts the approximate amortized computing cost for the Cloud MapReduce applications, with AzureMapReduce showing an advantage.

## VI. SUSTAINED PERFORMANCE OF CLOUDS

When discussing about cloud performance, the sustained performance of the clouds is often questioned. This is a valid question, since clouds are often implemented using a multi-tenant shared VM-based architecture. We performed an experiment by running the SWG EMR and SWG AzureMapReduce using the same workload throughout

different times of the week. In these tests, 32 cores were used to align 4000 sequences. The results of this experiment are given in Figure 5. Each of these tests was performed at +/- 2 hours 12AM/PM. Figure 5 also includes normalized performance for AzureMapReduce, calculated using the EMR as the baseline. We are happy to report that the performance variations we observed were very minor, with standard deviations of 1.56% for EMR and 2.25% for AzureMapReduce. Additionally, we did not notice any noticeable trends in performance fluctuation.

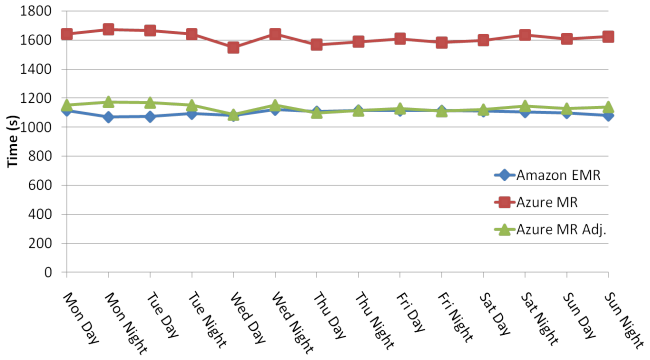


Figure 5. Sustained performance of cloud environments

## VII. CONCLUSION

We introduced the novel decentralized controlled AzureMapReduce framework, which fulfills the much-needed requirement of a distributed programming framework for Azure users. AzureMapReduce is built using Azure cloud infrastructure services that take advantage of the quality of service guarantees provided by the cloud service providers. Even though cloud services have higher latencies than their traditional counter parts, scientific applications implemented using AzureMapReduce were able to perform comparatively with the other MapReduce implementations, thus proving the feasibility of AzureMapReduce architecture. We also explored the challenges presented by cloud environments to execute MapReduce computations and discussed how we overcame them in the AzureMapReduce architecture.

We also presented and analyzed the performance of two scientific MapReduce applications on two popular cloud infrastructures. In our experiments, scientific MapReduce applications executed in the cloud infrastructures exhibited performance and efficiency comparable to the MapReduce applications executed using traditional clusters. Performance comparable to in house clusters, on demand availability, horizontal scalability and the easy to use programming model together with no upfront cost makes using MapReduce in cloud environments a very viable option and an enabler for the computational scientists, especially in scenarios where in-house compute clusters are not readily available. From an economical and maintenance perspective, it even makes sense not to procure in-house clusters if the utilization would be low. We also observed that the fluctuation of cloud MapReduce performance is minimal over a weeklong period, assuring consistency and predictability of application performance in the cloud environments.

## ACKNOWLEDGMENT

We would like to thank all the SALSA group members for their support. We appreciate Microsoft for their technical support on Azure. This work was made possible using the compute use grant provided by Amazon Web Service which is titled "Proof of concepts linking FutureGrid users to AWS". This work is partially funded by Microsoft "CRMC" grant and NIH Grant Number RC2HG005806-02.

## REFERENCES

[1] T. Hey, S. Tansley, and K. Tolle, *Jim Gray on eScience: a transformed scientific method*: Microsoft Research, 2009.

[2] T. Gunarathne, T.-L. Wu, J. Qiu *et al.*, "Cloud computing paradigms for pleasingly parallel biomedical applications," in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, Illinois, 2010, pp. 460-469.

[3] J. Dean, and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM,* vol. 51, no. 1, pp. 107-113, 2008.

[4] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for Data Intensive Scientific Analyses." pp. 277-284.

[5] Qiu X., Ekanayake J., Gunarathne T. *et al.*, "Using MapReduce Technologies in Bioinformatics and Medical Informatics."

[6] C. Evangelinos, and C. N. Hill, "Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2.," in Cloud computing and it's applications (CCA-08), Chicago, IL, 2008.

[7] *Apache Hadoop*, Retrieved Aug 20, 2010, from ASF: http://hadoop.apache.org/core/.

[8] *Amazon ElasticMapReduce*, Retrieved Aug 20, 2010, http://aws.amazon.com/elasticmapreduce/.

[9] *cloudmapreduce*, Retrieved Aug 20, 2010: http://sites.google.com/site/huanliu/cloudmapreduce.pdf.

[10] T. F. Smith, and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology,* vol. *147*, no. *1*, pp. 195-197, 1981.

[11] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology,* vol. 162, pp. 705-708, 1982.

[12] X. Huang, and A. Madan, "CAP3: A DNA sequence assembly program," *Genome Res,* vol. 9, no. 9, pp. 868-77, 1999.

[13] "*Hadoop Distributed File System HDFS*," December, 2009; http://hadoop.apache.org/hdfs/.

[14] *Amazon Web Services*. http://aws.amazon.com

[15] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu *et al.*, "Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications," in Proceedings of the Emerging Computational Methods for the Life Sciences Workshop of ACM HPDC 2010 conference, Chicago, Illinois, 2010.

[16] *Windows Azure Platform*, Retrieved April 20, 2010, from Microsoft: http://www.microsoft.com/windowsazure/.

[17] "MPI," *Message Passing Interface*, http://www-unix.mcs.anl.gov/mpi/, 2009].

[18] J.Ekanayake, H.Li, B.Zhang *et al.*, "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, Chicago, Illinois, 2010.

[19] J. Ekanayake, T. Gunarathne, J. Qiu *et al.*, "Cloud Technologies for Bioinformatics Applications," *Accepted for publication in Journal of IEEE Transactions on Parallel and Distributed Systems*, 2010.

[20] *AppEngine-MapReduce*, vol. 2010, Retrieved September 2, 2010, http://code.google.com/p/appengine-mapreduce/.

[21] "JAligner.," December, 2009; http://jaligner.sourceforge.net.