**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*— Utility computing model introduced by cloud computing together with the rich set of cloud infrastructure services offers a very viable alternative for the traditional servers and compute 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 the cloud environments such as using MapReduce as a service, setting up your own MapReduce cluster on cloud instances as well as using specialized cloud MapReduce runtimes which take advantage of the cloud infrastructure services. In this paper we introduce AzureMapReduce, a novel MapReduce runtime built using the Microsoft Azure cloud infrastructure services. Then we evaluate the use and performance of MapReduce, including AzureMapReduce, in the cloud environments for scientific applications using sequence assembly and sequence alignment as use cases.

Keywords-component; formatting; style; styling; insert (key words)

#  Introduction

Today, the scientific discovery is increasingly relying on processing of very large amounts of data, which Jim Gray coined as the 4th paradigm [1]. The scientists are relying on computational resources more than ever and always crave for more power and ease of use. At the same time we notice the introduction of the concepts of Cloud Computing and MapReduce by the industry giving very viable alternatives for the computational needs of the scientists. In fact the utility computing model offered by cloud computing is very well suited for the bursty compute needs of the scientists. While clouds offer raw compute power together with infrastructure service offering storage and other services, there is a need for distributed computing frameworks to harness the power of clouds 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.

MapReduce distributed data analysis framework model introduced by Google [2] provides an easy to use programming model together with fault tolerance, automatic parallelization, scalability and data locality based optimizations. MapReduce frameworks are well suited to execute large distributed jobs in brittle environments such as commodity clusters and cloud infrastructures, due to its excellent fault tolerance features. Though introduced by the industry and used mainly in the information retrieval community, it is shown that MapReduce frameworks are a capable of supporting many scientific application use cases, making it a good choice for scientists to easily build large data intensive applications that needs to be executed in the cloud infrastructures.

There exist many options to execute MapReduce jobs on the cloud environments, such as manually setting up a MapReduce(eg: Hadoop [3]) cluster on a leased set of compute instances, using an on demand MapReduce-as-service offering such as Amazon ElasticMapReduce(EMR) [4] or using a cloud MapReduce runtime such as AzureMapReduce or CloudMapReduce [5]. In this paper we explore and evaluate these different options for three well known bioinformatics applications, SmithWatermanGOTOH pairwise distance alignment (SWG) [6-7], Cap3 [8] sequence assembly and BLAST[9]. Also we perform experiments to gain an insight about the performance of MapReduce in the clouds for the selected applications and compare with MapReduce on traditional clusters.

Our work was motivated by an experience we had in early 2010 when we evaluated the use of Amazon EMR for our scientific applications. To our surprise, we observed subpar performance in EMR than using a manually build cluster on EC2, which prompted us to perform the current analyses. In this paper we show that Map Reduce in the clouds has the ability to perform comparably to MapReduce clusters on dedicated private clusters.

Microsoft Azure platform currently do not provide or support any distributed parallel computing frameworks such as MapReduce, Dryad or MPI other than using the Azure queue based job scheduling. Also the platform as a service nature of Azure makes it hard or rather impossible to setup an existing general purpose runtime on 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 services. In this paper we use an experimental version of AzureMapReduce for our experiments.

# Technologies

##  Hadoop

Apache Hadoop [3] MapReduce is an open source MapReduce style distributed data processing framework, which is an implementation similar to the Google MapReduce [2]. Apache Hadoop MapReduce uses HDFS [10] distributed parallel file system for data storage, which stores the data across the local disks of the compute 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 replication of file data. When executing Map Reduce programs Hadoop optimize the data communication by scheduling computations near the data by using the data locality information provided by the HDFS file system. Hadoop follows a master node with many client workers architecture and uses a global queue for the task scheduling, 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 execution of slower tasks and handles failures by rerunning of the failed tasks using different workers.

## Amazon Web Services

Amazon Web Services (AWS) [11] are a set of on demand over the wire (online) cloud computing services offered by Amazon. AWS offers a wide range of compute, 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).

Amazon EC2 service enables the users to lease hourly billed Xen based virtual machine instances over the internet 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, where the users gets access directly 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 compute capacity, memory and the I/O performance with differing price points, giving users the flexibility to be economical by choosing the best matching instances for the use case. In [12] we perform an analysis about a select set of EC2 instance types. EC2 offers both Linux instances as well as Windows instances.

## 2.3 Amazon Elastic Map Reduce

Amazon Elastic MapReduce(EMR) [11] provides MapReduce as an on-demand service hosted in the Amazon infrastructure. EMR is a hosted Apache Hadoop [3] MapReduce framework, which utilizes Amazon EC2 for compute 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 a 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, EMR provides Bootstrap actions that users can specify to run on the compute nodes prior to the launch of the MapReduce job, which can be used for optional custom preparation of the images. EMR allows debugging of EMR jobs by providing an 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 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 the saving of files in native formats in S3.

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

## Microsoft Azure Platform

Microsoft Azure platform [13] is a cloud computing platform offering 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 in Amazon EC2, in Azure users do not get 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 more capability to offer quality of service and automated management services than in infrastructure-as-a-service offerings. Azure offers a limited set of instances on a linear price and feature scale.

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 an application running in the cloud. Messages can only contain text data and the size is limited to 8KB per message. Users can create unlimited number of queues and send unlimited number of messages. Azure queue does not guarantee the order of the messages, the deletion of messages and availability of all the messages for a single request, though it guarantees the eventual availability over multiple requests. Each message has a configurable visibility timeout. Once it’s read by a client, the message will not be visible for other clients till the visibility time expires. Message will reappear upon expiration of the timeout, as long as the previous reader did not delete it.

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.


# Science applciations Using MapReduce

What applications are better suited for MR (pleasingly parallel & loosely asynchronous?)

Why we selected the particular applications (bioinformatics has the need and the potential, etc)

# Challenges for MapREduce in the Clouds

* Data storage
	+ In cloud storage (eg:S3) or block storage (EBS) or just use VM storage to setup HDFS like file systems
* Metadata storage
	+ Single point of failures?
* Communication consistency & scalability
	+ I/O performance fluctuations
* Performance consistency (sustained performance) – Shared infrastructure, VM’s, Load based on time & date) etc
* Reliability (eg: Node failures)
	+ For both during runs as well as the burden of maintainance
* Choosing correct Virtual machine configurations (memory, CPU, IO, etc)
* Access to different virtual machines (ssh key based)

# AzureMapReduce

AzureMapReduce is a distributed decentralized MapReduce runtime for Windows Azure developed using Azure cloud infrastructure services. Usage of 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 avoiding single point of failures, bandwidth (network as well as storage) bottlenecks and management overheads. In this paper, we use a pre-release experimental version of AzureMapRedcue which we plan to release to the general public soon.

1. AzureMapReduce Architecture

The usage of cloud services usually introduce latencies larger than their optimized non-cloud counterparts and often do not guarantee the time for first availability of data. These overheads can be overcome by using sufficiently coarser grained map and reduce tasks. AzureMapReduce overcome the availability issues with the use of retrying and by designing the system not to rely on 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/reduce task scheduling, Azure tables for metadata & monitoring data storage, Azure blob storage for input/output/intermediate data storage and Window Azure Compute worker roles to perform the computations.

Google MapReduce [2], Hadoop [3] as well as Twister [14] 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 among other responsibilities. Cloud environments by design are more brittle than the traditional compute clusters and the cloud applications should be developed to expect and withstand the failures. Hence it is not possible for AzureMapReduce to make the same reliability assumption 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 avoiding the possible single point of failure. AzureMapReduce also provides the users with the capability to dynamically scale up or down the number of compute instances, even in middle of a MapReduce computation, as and when needed. The map and reduce tasks of the AzureMapReduce runtime are dynamically scheduled using a global queue. As we explored in one of our previous works [15], we experimentally showed that dynamic scheduling through a global queue achieves better load balancing across the tasks resulting in better performance and throughput than statically scheduled runtimes, especially when used with real world inhomogeneous data distributions.

## 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 automatically based on a data set present in the Azure Blob storage or manually based on custom criteria, which we find as a very useful feature when implementing science applications using MapReduce. Client driver uses the .net task parallel library to dispatch tasks parallel to overcome the latencies of the Azure queue and the Azure table services. Client driver can optionally be used to monitor the progress and the completion of the MapReduce jobs.

## 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 contains meta-data needed for the Map task execution, such as input data file location, program parameters, map task ID, etc.. 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. We are actively working on investigating other approaches for the intermediate data transfer.

## Reduce Tasks

Reduce task scheduling is similar to the 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 starts as soon as the first Map Task completion, overlapping the data transfer with the computation. This overlapping of data transfer with computation minimizes the data transfer overhead of the MapReduce computations, which was evident 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 the that reduce task is fetched. In the AzureMapReduce, each reduce task will independently determine (decentralized control) the completion of map tasks based on the information in the map task meta-data table and the reduce task meta-data table. Reduce tasks uploads the results to the Azure Blob Storage.

## 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’s possible to monitor the intermediate data transfer progress using the tables for each reduce task. A monitoring web application (Azure web role) to dynamically display the contents of these tables is currently under the development.

##  Fault Tolerance

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

## Limitations of Azure

Queue time out

Aggregate count support in tables

## Related technologies

### Google AppEngine-MapReduce: Google AppEngine-MapReduce [16] 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 libaray, while supporting Reduce phase is part of the 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 executable, or threads.

### Cloud map reduce: CloudMapReduce [5] also implements a decentralized MapReduce architecture using the cloud infrastructure services of Amazon Web Services. The main differences between CloudMapReduce and AzureMapReduce pre-release version lies in the method of handling the intermediate data & meta-date and the timing of the starting of the reduce tasks among others.

# Performance of MapReduce in the clouds for science

## Methodology

We performed scalability tests for each of the application to evaluate the performance of the MapReduce implementations in the cloud runtimes 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 remains constant.

All the AzureMapReduce tests were performed using Azure small instances (1 CPU core). The bare metal tests were performed on an iDataplex cluster with each node having two 4-core CPUs (Intel Xeon CPU E5410 2.33GHz) and 16 GB memory, and connected using Gigabit Ethernet network interface. The EC2 and EMR tests for Cap3 and Blast applications 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 get 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 get billed as 8 Amazon compute 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, even though an extra compute node was used as the Hadoop master node.

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

Below defined parallel efficiency and relative parallel efficiency calculations are used when presenting the results.

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 are present in the local disks. T(ρ) is the parallel run time for the application while “p” is the number of processor cores used.

We calculate the relative parallel efficiency when estimating the sequential run time for an application is not straightforward. α = *p/p*1, where p1 is the smallest number of CPU cores for the experiment.


##  Smith-Waterman-GOTOH(SWG) pairwise distance calculation

In this application we use Smith-Waterman [6] algorithm with GOTOH [7] (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 all-pairs dissimilarity for a data set, calculating only the strictly lower or upper 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 2, 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 smaller, while the size of the intermediate and the output data are significantly larger due to the n2 result space.

More details about the Hadoop-SWG application implementation is given in [15]. The AzureMapReduce implementation also follows the same architecture and the blocking strategy as in the Hadoop-SWG implementation. Hadoop-SWG uses the open source JAligner [17] as the computational kernel, while AzureMapReduce SWG uses the .net implementation, NAligner [17] as the computational kernel. The results of the SWG MapReduce computation gets 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 happens linearly. The pure performance results of the SWG MapReduce scalability test is given in figure 3(a). A block size of 200 \* 200 sequences is used in the performance experiments resulting in 40,000 sequence alignments per block, which results in ~123 million sequence comparisons in the 3072 block test case.

1. SWG MapReduce task decomposition

Due to the sheer size of even the smallest computation in our SWG scaling test, we found it impossible to calculate the sequential time. Also due to the all-pairs nature of SWG, it’s not possible to calculate the sequential running 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 moderate sized sequential SWG calculation in all the environments and used that result to normalize the performance using bare metal performance as the baseline. The normalized performance is depicted in figure 3(c), where we can observe that all four applications show comaparable performance for SWG.

In figure 3(d) we present the approximate computational costs for the experiments performed using the cloud infrastructures. Costs presented here are amortized for the actual time (time / 3600 \* num\_instances \* instance\_price\_per\_hour), assuming the remaining time of the instance hour have been put to useful work. In addition to this, there will be minor charges for the data storage for EMR & AzureMapReduce. Also there will be additional minor charges for the queue service and table service for AzureMapReduce. We can notice that the costs for Hadoop on EC2 and AmazonMapReduce are similar, while EMR costs a fraction more.

## Sequence assembly using Cap3

Cap3 [8] 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 presented as FASTA formatted files generates consensus sequences. Cap3 program is often used in parallel with lots of input files due to the pleasingly parallel nature of the application. Size of a typical data input file for Cap3 program and the result data file range from hundreds of kilobytes to few megabytes. Output files resulting from the input data 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 [12].

We used a replicated set of Fasta files as the input data in our experiments. Each of the file contained 458 reads. Figure 5(a) presents the pure performance numbers for the Cap3 MapReduce applications, while figure 5(b) presents the absolute parallel efficiency for the Cap3 MapReduce applications. As we can notice, all the cloud Cap3 applications displayed performance comparative to the bare metal clusters, while AzureMapReduce and Hadoop Bare metal showed a slight edge over the Amazon counterparts in terms of the efficiency. Figure 5(c) depicts the approximate amortized compute cost for the Cloud MapReduce applications, with AzureMapReduce showing an advantage

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

## Sequence alignment using BLAST

NCBI BLAST+ [9] is a very popular bioinformatics standalone application to handle sequence similarity searching. It is the latest version of BLAST [18], a multi-letter command line tool developed using NCBI C++ toolkit, to translate a FASTA formatted nucleotide query and to compare it to a protein database.

For BLAST+ execution, the application requires few important parameters, such as a query and database. The database is normally large up to gigabytes. Although BLAST+ is much faster than original BLAST (written in C) with multi-thread support, the performance is still a problem when increasing the query size.

In this paper, for our Hadoop implementation we follow an approach similar to CloudBLAST [19] which utilizes Hadoop MapReduce. The database is distributed using distributed cache, which gets duplicated to the local disk of each slave (worker) to serve for the entire job. Then, for each map tasks, it downloads and processes an assigned query from HDFS and uploads the results back to HDFS. For the AzureMapReduce, we used the data preloading feature of AzureMapReduce, which similar to Hadoop distributed cache downloads a specified set of files to all the compute nodes before starting the computation, to distribute the Blast database. A map only program similar to Cap3 is used to implement the AzureMapReduce Blast application. Figure 6 depicts the performance of Blast MapReduce applications. We selected a set of query files to be used as the smallest test case and then replicated it proportionally to number of cores to perform this experiment. We can notice the Blast performance to be more varied than our previous applications.

1. Performance of Blast MapReduce applications

# Sustained performance of clouds

When we talk about the cloud performance, a question that gets raised often is about the sustained performance of the clouds. This is a valid question since often clouds are 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 are used to align 4000 sequences. The results of this experiment are given in the figure 4.Each of these tests were performed +/- 2 hours 12AM/PM. Figure 4 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. Also we did not notice any noticeable trends in the performance fluctuations.

1. Sustained performance of cloud environments

# Conclusion

In this paper we presented and analyzed the performance of scientific MapReduce applications on the 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, further confirming the viability of MapReduce in cloud infrastructure for the data intensive scientific analyses. We also observed that the fluctuation of cloud performance is minimal over a period of a week by performing experiments in different times of the week.

We also introduced the novel AzureMapReduce framework, which is built using Azure cloud infrastructure services. Even though cloud services have higher latencies than their traditional counter parts, scientific applications implemented using AzureMapReduce was able to perform comparatively with the other MapReduce implementations, proving the feasibility of our AzureMapReduce architecture.

##### Acknowledgment

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression, “One of us (R.B.G.) thanks . . .” Instead, try
“R.B.G. thanks”. Put applicable sponsor acknowledgments here; DO NOT place them on the first page of your paper or as a footnote.

##### References

1. [1] T. Hey, et al., Jim Gray on eScience: a transformed scientific method: Microsoft Research, 2009.
2. [2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Commun. ACM, vol. 51, pp. 107-113, 2008.
3. [3] "Apache Hadoop," Retrieved April 20, 2010, from ASF: <http://hadoop.apache.org/core/>.
4. [4] "Amazon ElasticMapReduce," <http://aws.amazon.com/elasticmapreduce/>.
5. [5] "cloudmapreduce," Retrieved April 20, 2010: <http://code.google.com/p/cloudmapreduce/>.
6. [6] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," Journal of Molecular Biology, vol. 147, pp. 195-197, 1981.
7. [7] O. Gotoh, "An improved algorithm for matching biological sequences," Journal of Molecular Biology, vol. 162, pp. 705-708, 1982.
8. [8] X. Huang and A. Madan, "CAP3: A DNA sequence assembly program," Genome Res, vol. 9, pp. 868-77, 1999.
9. [9] G. C. Christiam Camacho, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer and Thomas L Madden, "BLAST+: architecture and applications," BMC Bioinformatics 2009, 10:421, 2009.
10. [10] (2009, December). Hadoop Distributed File System HDFS. Available: <http://hadoop.apache.org/hdfs/>
11. [11] Amazon, "Amazon Web Services."
12. [12] Thilina Gunarathne, et al., "Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications," presented at the Proceedings of the Emerging Computational Methods for the Life Sciences Workshop of ACM HPDC 2010 conference, Chicago, Illinois, 2010.
13. [13] "Windows Azure Platform," Retrieved April 20, 2010, from Microsoft: <http://www.microsoft.com/windowsazure/>.
14. [14] J.Ekanayake, et al., "Twister: A Runtime for iterative MapReduce," presented at the Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010, Chicago, Illinois, 2010.
15. [15] J. Ekanayake, et al., "Cloud Technologies for Bioinformatics Applications," Accepted for publication in Journal of IEEE Transactions on Parallel and Distributed Systems, 2010.
16. [16] "AppEngine-MapReduce," Retrieved September 2, 2010, <http://code.google.com/p/appengine-mapreduce/>.
17. [17] (2009, December). JAligner. Available: <http://jaligner.sourceforge.net>
18. [18] NCBI. (2010, BLAST. Available: <http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastNews#1>
19. [19] A. Matsunaga, et al., "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications," in IEEE Fourth International Conference on eScience (eScience '08), Indianapolis, IN, 2008.