Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications

Thilina Gunarathne[1,2], Tak-Lon Wu[1,2], Judy Qiu[2], Geoffrey Fox[1,2]
[1]School of Informatics and Computing, [2]Pervasive Technology Institute
Indiana University, Bloomington.
{tgunarat, taklwu, xqiu,gcf}@indiana.edu

ABSTRACT

Cloud computing offers new approaches for scientific computing that leverage the major commercial hardware and software investment in this area. Closely coupled applications are still unclear in clouds as synchronization costs are still higher than on optimized MPI machines. However loosely coupled problems are very important in many fields and can achieve good cloud performance even when pleasingly parallel steps are followed by reduction operations as supported by MapReduce. However we can use clouds in several ways and here we compare four different approaches using two biomedical applications. We look at the cloud infrastructure service based virtual machine utility computing models of Amazon AWS and Microsoft Windows Azure; Map Reduce based computing frameworks Apache Hadoop (deployed on raw hardware as well as on virtual machines) and Micrsoft DryadLINQ. We compare performance showing strong variations in cost between different EC2 machine choices and comparable performance between the utility computing (spawn off a set of jobs) and managed parallelism (MapReduce). The MapReduce approach offered the most user friendly approach.

General Terms

Measurement, Performance, Economics.

Keywords

Cloud Technology, Map Reduce, Interpolation, Sequence Assembly

1. INTRODUCTION

Scientists are overwhelmed with the increasing amount of data processing needs arising from the storm of data that is flowing through virtually every field of science. One example is the production of DNA fragments at massive rate by the now widely available automated DNA Sequencer machines. Cloud computing offerings by major commercial players provide on demand computational services over the web which can be purchased within minutes just by use of a credit card. This utility computing model offered through cloud computing opens up exiting new opportunities for the computational scientists to perform their computations, as this model suits well with the occasional resource intensive (spiky) compute needs of the scientists. Another advantage is the ability increase the throughput of the computation by horizontally scaling the compute resource without additional cost overhead as for an example 100 hours in 10 nodes cost same as 10 hours in 100 nodes.

In addition to the leasing of virtualized compute nodes, the cloud computing platforms offer a rich set of distributed cloud infrastructure services including storage, messaging and database services with cloud specific service guarantees. These services can be leveraged to build and deploy scalable distributed applications on cloud environments. At the same time we notice the emergence of different cloud oriented data processing frameworks such as Map Reduce[1] technologies, which allow users to effectively to perform distributed computations in increasingly brittle environments like commodity clusters and computational clouds.

In this paper we introduce a set of abstract frameworks constructed using the cloud oriented programming models to perform pleasingly parallel computations. We present implementations of bio medical applications such as Cap3[2] sequence assembly, MDS & GTM interpolation using these frameworks. We analyze the performance and the usability of different cloud oriented programming models using the Cap3 application to assemble a large collection of genome fragments and using GTM & MDS interpolation applications to process 26 million 166-dimensional dataset obtained from the PubChem project database. Pubchem is a NIH funded repository of over 60 million chemical molecules and provides their chemical structures and biological activities. We use Amazon Web Services[3] and Microsoft Windows Azure[4] cloud computing platforms and use Apache Hadoop[5] Map Reduce and Microsoft DryaLINQ[6] as the distributed parallel computing frameworks.

2. CLOUD TECHNOLOGIES AND APPLICATION ARCHITECTURE

Processing of large data sets using existing sequential executables is a common use case we encounter in many scientific applications. Some of these applications exhibit pleasingly parallel characteristics where the data can be independently processed in parts allowing the applications to be easily parallelized. In the following sections we explore the cloud programming models and application frameworks we developed using them for the pleasingly parallel computations. These frameworks have been used to implement the applications mentioned in section 3.

2.1 Amazon Web Services & Microsoft Azure Platform

Amazon Web Services (AWS)[3] are a set of cloud computing services by Amazon, offering on demand compute and storage services including but not limited to Elastic Compute Cloud (EC2), Simple Storage Service (S3) and Simple Queue Service (SQS). EC2 provides users the capability to lease hourly billed Xen based virtual machine instances which allows users to dynamically provision resizable virtual clusters in a matter of minutes through a web service interface. EC2 supports both Linux and Windows virtual instances. SQS is an eventual consistent distributed message queue service which provides a reliable, scalable messaging mechanism for storing messages while communicating with distributed components of an application. S3 is a distributed storage system where users can store and retrieve data through a web services interface and is accessible from anywhere in the web.

Amazon EC2 offers a variety of instance types giving a richer set of options for the user to choose from depending on his need. One particular instance type of interest is the high CPU extra large instances, which costs the same as an extra large instance but offers more CPU power at the cost of lesser memory. Similarly EC2 offers high-memory instance types too. Table 1 provides a summary of the EC2 instance types we used in this paper.

Table 1 : Selected EC2 Instance Types

Instance Type	Memory	EC2 compute units	Actual CPU cores	Cost per hour
Large (L)	7.5 GB	4	2 X (~2Ghz)	0.34$
Extra Large (XL)	15 GB	8	4 X (~2Ghz)	0.68$
High CPU Extra Large (HCXL)	7 GB	20	8 X (~2.5Ghz)	0.68$
High Memory 4XL (HM4XL)	68.4 GB	26	8X (~3.25Ghz)	2.40$

Microsoft Azure platform[4] also offers a set of cloud computing services similar to the Amazon web services. Windows Azure compute, Azure Storage Queues and Azure Storage blob services are the Azure counterparts for Amazon EC2, Amazon SQS and the Amazon S3 services. Azure only supports Windows virtual instances and offer a limited variety of instance types when compared with Amazon EC2. A single Azure small instance costs 0.12$ per hour and comprises of 1 CPU core, 1.7 GB memory and 250 GB disk space. Azure medium, large and extra large instances are multiples of 2, 4 & 8 of the small instance in cost wise as well as in the configuration wise. It's been speculated that a single CPU core in Azure is 1.5Ghz to 1.7Ghz. In section 4.1 for the Cap3 program, we found that 8 Azure small instances perform as better as a single Amazon high CPU extra large instance with 20 compute units.

2.1.1 *Classic cloud architecture*

Figure 1 depicts the architecture of the classic cloud processing model. The classic cloud processing model uses the cloud instances (EC2 or Azure) for data processing and uses Amazon S3 or Windows Azure for the data storage. For the task scheduling, it uses an Amazon SQS or an Azure queue as a queue of tasks where every message in the queue corresponds to a single task. The client populates the scheduling queue with tasks, while the worker processes pick tasks from the scheduling queue. Both SQS queue instances and the Azure queue instances have a configurable visibility timeout where a message will not be visible to other workers for the given amount of time after it's read by a worker unless it is deleted by the worker upon completion of the task. This feature provides a simple fault tolerance capability to the system, where a message will get processed by some worker if the task does not get completed with the initial reader (worker) within a certain time limit.

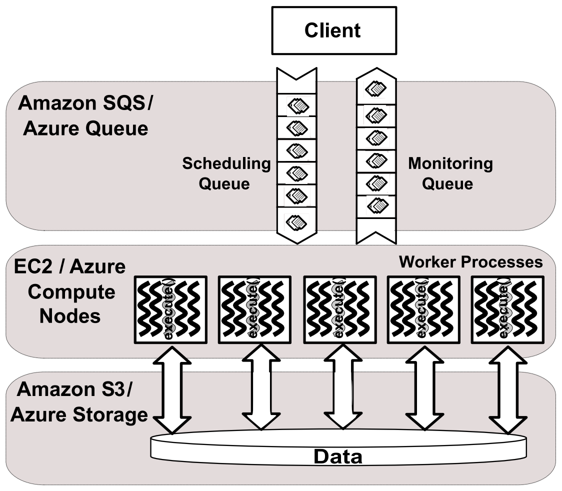

Figure 1: Classic Cloud Processing Model

In the applications discussed in this paper a single task corresponds to a single input file. The worker processes will retrieve the input files from the cloud storage and will process them using an executable program before uploading the results back to the cloud storage. In this implementation a user can configure the workers to use any executable program installed in the virtual machine provided that it takes input in the form of a file. Our implementation uses a monitoring message queue to monitor the progress of the computation, but for more sophistication one can use cloud data stores like Amazon simpleDB to store the monitoring and state data.

2.2 Hadoop

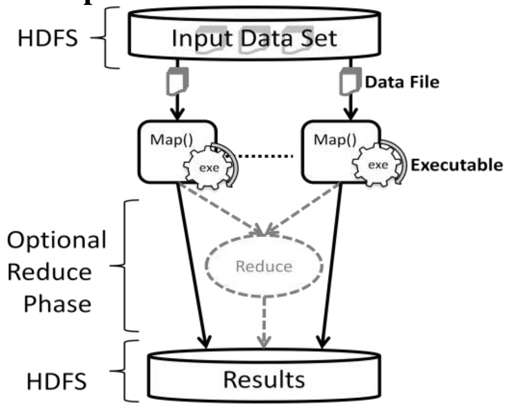

Figure 2: Hadoop Map Reduce Processing Model

Apache Hadoop[5] is an open source implementation of the Google Map Reduce[1] technology. It shares many characteristics with the Google MapReduce implementation. Apache Hadoop uses HDFS 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. Hadoop optimizes the data communication by scheduling computations near the data using the data locality information provided by the HDFS file system. Hadoop handles failures by rerunning the failed tasks.

The parallel application framework on Hadoop is developed as a set of map tasks which process the given data splits using the configured executable program. Input to Hadoop map tasks comprises of key, value pairs, where by default Hadoop parses the contents of the file split to read them. Frequently the executable data processing programs expect a file path for the input, rather than the contents, which is not possible with the Hadoop built-in formats and record readers. We implemented a custom InputFormat and a RecordReader for Hadoop so that map tasks will receive the file name and the HDFS path of the data split respectively as the key and the value, while preserving the Hadoop data locality based scheduling.

2.3 DryadLINQ

Dryad[7] is a framework developed by Microsoft Research as a general-purpose distributed execution engine for coarse-grain parallel applications. Dryad applications are expressed as directed acyclic data-flow graphs (DAG), where vertices represent computations and edges represent communication channels between the computations. Similar to Map Reduce, Dryad scheduler optimizes the data transfer overheads by scheduling the computations near data and handles failures through rerunning of tasks and duplicate instance execution. Data for the computations need to be partitioned manually and stored beforehand in the local disks of the computational nodes via windows shared directories. Dryad is available for academic usage through the DryadLINQ API. DryadLINQ[8] is a high level declarative language layer on top of Dryad. DryadLINQ queries get translated in to distributed Dryad computational graphs in the run time. Latest version of DryadLINQ operates on Window HPC clusters only.

The DryadLINQ implementation of the framework uses the DryadLINQ "select" operator on the data partitions to perform the distributed computation. The resulting computation graph looks much similar to the figure 2, where instead of using HDFS, Dryad will use the windows shared local directories for data storage. As mentioned above, the data partitioning, distribution and the generation of metadata files for the data partitions needs to be performed manually.

2.4 Usability of the technologies

As expected, implementing the above mentioned application framework using Hadoop and DryadLINQ was easier than implementing them using the cloud services as most of the data processing framework was already in place with Hadoop & DryadLINQ. Hadoop & DryadLINQ takes care of scheduling, monitoring and fault tolerance. With Hadoop we had to implement a Map function, which contained the logic to copy the input file from HDFS to the working directory, execute the external program as a process and to upload the results files to the HDFS. In addition to this, we had to implement a custom InputFormat and a RecordReader to support file inputs to the map tasks. With DryadLINQ we had implement a side effect free function to execute the program on the given data and to copy the result to the output shared directory. But significant effort had to be spent on implementing the data partitioning and the distribution programs to support DryadLINQ.

EC2 & Azure classic cloud implementations involved more effort than the Hadoop & DryadLINQ implementations, as all the scheduling, monitoring and fault tolerance had to be implemented from the scratch using the features of the cloud services. Amazon EC2 provides infrastructure as a service by allowing users to access the raw virtual machine instances while windows Azure provides the .net platform as a service allowing users to deploy .net applications in the virtual machines through a web interface. Hence the deployment process was easier with Azure as oppose to the EC2 where we had to manually create instances, install software and start the worker instances. On the other hand the EC2 infrastructure as a service gives more flexibility and control to the developers. Azure provides better development & testing support through the visual studio integration and the local debugging environment of the Azure SDK. Azure platform is heading towards providing a more developer friendly environment, but as of today (Mar 2010) the platform is less mature compared to the AWS.

3. APPLICATIONS

3.1 Cap3

Cap3 [2] is a sequence assembly program which assembles DNA sequences by aligning and merging the sequence fragments from the respective sequences. Sequence assembly is an integral part of genomics as the current DNA sequencing technology, such as shotgun sequencing, is capable of reading only parts of whole genomes at once. The Cap3 algorithm operates on collection of gene sequence fragments presented as FASTA formatted files. It removes the poor regions of the DNA fragments, calculates the overlaps between the fragments, identifies and removes the false overlaps, joins the fragments to form contigs of one or more overlapping DNA segments and finally through multiple sequence alignment generates consensus sequences.

The increased availability of DNA Sequencers is generating massive amounts of sequencing data that needs to be assembled. Cap3 program is often used in parallel with lots of input files due to the pleasingly parallel nature of the application.

3.2 GTM & MDS Interpolation

MDS[9] and GTM[10] are known as an algorithm for dimension reduction, which finds an optimal user-defined low-dimensional representation out of the data in high-dimensional space. However, although MDS and GTM share the same objective for optimal dimension reduction, GTM finds a non-linear mapping based on Gaussian probability density model in vector space. On the other hand, MDS tries to construct a mapping in target dimension with respect to the pairwise proximity information, mostly dissimilarity or distance.

Multidimensional Scaling (MDS): MDS is a general term of the techniques to configure low dimensional mappings of the given high-dimensional data with respect to the pairwise proximity information, while the pairwise Euclidean distance within target dimension of each pair is approximated to the corresponding original problem to find low-dimensional configuration which minimizes the objective function, called STRESS or SSTRESS. Definition of STRESS (1) and STRESS (2) are as following:

$$\sigma(X) = \sum_{i<j\leq N} w_{ij}\left(d_{ij}(X) - \delta_{ij}\right)^2 \qquad (1)$$

$$\sigma^2(X) = \sum_{i<j\leq N} w_{ij}\left(d_{ij}^2(X) - \delta_{ij}^2\right)^2 \qquad (2)$$

where $d_{ij}(X) = ||x_i - x_j||$, $1 \leq i < j \leq N$ and w_{ij} is a weight value, so $w_{ij} \geq 0$.

Generative Topographic Mapping (GTM): GTM is an unsupervised learning method for modeling the density of data and finding a non-linear mapping of high-dimensional data in a low-dimensional space. GTM is also known as a principled alternative to Self-Organizing Map (SOM) [11]which does not have any density model, GTM defines an explicit density model based on Gaussian distribution [12] and finds the best set of parameters associated with Gaussian mixtures by using an Expectation-Maximization (EM) optimization algorithm[13].

Interpolation of MDS[14] and GTM is an out-of-sample extension of the original algorithms and it is designed to process much larger data points with minor trade-off of approximation. Instead of processing full dataset which is the case of original MDS and GTM algorithms, interpolation approach takes only a part of the full dataset, known as samples, for a computer-intensive training process and applies the trained result to the rest of dataset, known as out-of-samples, which is usually faster than the former process. With this interpolation approach in MDS and GTM, one can visualize millions of data points with modest amount of computations and memory requirement. Currently we use MDS, GTM, MDS interpolation and GTM interpolation applications for DNA sequence studies and chemical information mining & exploration of the Pubchem database data.

4. PERFORMANCE

4.1 Application performance with different cloud instance types

Table 2: EC2 Cap3 performance with different instance types to assemble 200 FASTA files with 458 reads in each file

Node Type	No. of Nodes	No. of Workers per Node	Total Time (s)	Amortized Compute Cost	Compute Cost (hour units)
Large	10	2	1389.0	$2.33	$3.40
Large	10	4	1414.6	$1.34	$3.40
XLarge	5	2	2486.6	$2.35	$3.40
XLarge	5	4	1424.3	$1.35	$3.40
XLarge	5	8	1459.6	$1.38	$3.40
HCXL	2	5	2302.2	$0.87	$1.36
HCXL	2	10	1554.7	$0.59	$1.36
HCXL	2	20	1452.9	$0.55	$1.36

Table 2 presents the benchmark results for Cap3 application on different EC2 instance types. EC2 small instance size was not included in our study as it does not support 64bit operating systems. EC2 instances are hourly billed. The compute cost (hour units) assumes the instances are used only for that particular computation. The amortized cost assumes that the instance will be used for useful work for the remainder of the hour.

According to these results we can infer that memory is not a bottleneck for the Cap3 program and that for the above instance types, the performance is proportional to the number of actual compute cores. The most efficient performance for the Cap3 EC2 application is gained using high CPU extra large instances.

Table 3 : EC2 GTM performance with different instance types to process 100,000 Pubchem data points

Node Type	No. of Nodes	No. of Workers per Node	Total Time (s)	Amortized Compute Cost	Compute Cost (per hour units)
Large	5	2	478.8	$0.23	$1.70
Large	5	4	422	$0.20	$1.70
XLarge	2	5	710.8	$0.27	$1.36
XLarge	2	10	721.5	$0.27	$1.36
HCXL	1	10	866.5	$0.16	$0.68
HCXL	1	20	878.4	$0.17	$0.68
HM4XL	1	10	320.8	$0.21	$2.40
HM4XL	1	20	319.5	$0.21	$2.40

According to the table 3, we can infer that memory (size & bandwidth) is a bottleneck for the GTM interpolation application. The GTM interpolation application performs better in the presence of more memory and less processor core sharing the memory. But still the high CPU extra large instance appears as the most economic choice.

4.2 Performance comparison between different implementations

In the following studies we use parallel efficiency as the measure to evaluate the different frameworks. Efficiency is calculated using the following formula.

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

T(1) is the best sequential execution time for the application in a particular environment using the same data set or a representative subset if the sequential time is prohibitively large to measure. 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.

Efficiency is a relatively good measure to evaluate the different approaches we use in our studies as we don't have the possibility to use identical configurations across the different environments. At the same time we cannot use efficiency to directly compare the the different technologies due to the following reasons. Even though efficiency accounts the differences of the systems which effects the sequential running time as well as the parallel running time, it does not reflect other differences such as memory size, memory bandwidth and network that can effect when running parallel computations.

Per core per computation time is calculated in each test to give an idea about the actual execution times in the different environments.

$$Per\ Computation\ Per\ Core\ time = \frac{\mathrm{pT}(\rho)}{No.of\ computations}$$

4.2.1 Cap3

We benchmarked the Cap3 classic cloud implementation performance using a replicated set of FASTA formatted data files, each containing 458 reads, and compared with our previous performance results[15] for Cap3 DryadLINQ on bare metal, Cap3 Hadoop bare metal and Cap3 Hadoop on virtual machine results. 16 High CPU extra large instances were used for the EC2 testing and 128 small instances were used for the Azure Cap3 testing. DryadLINQ, Hadoop bare metal & Hadoop VM results were obtained using a 32 node X 8 core (2.5 Ghz) cluster with 16 GB memory in each node. An EC2 extra large instance was considered as 8 actual cores while an Azure small instance was considered as a single core for the following calculations. In all the cases the data was already present in the frameworks preferred storage location.

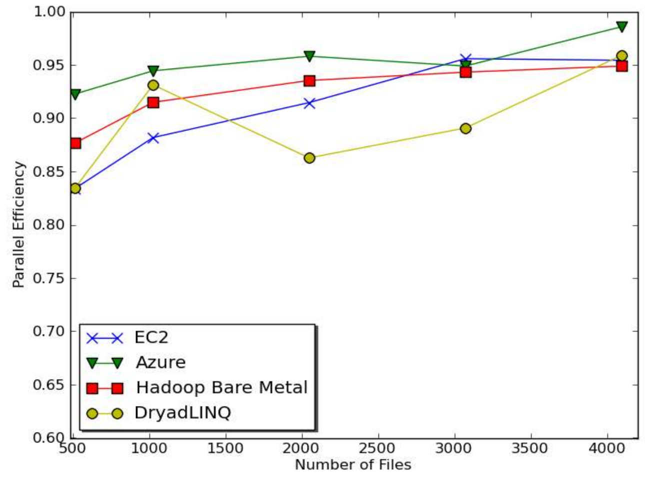

Figure 3 : Cap3 parallel efficiency

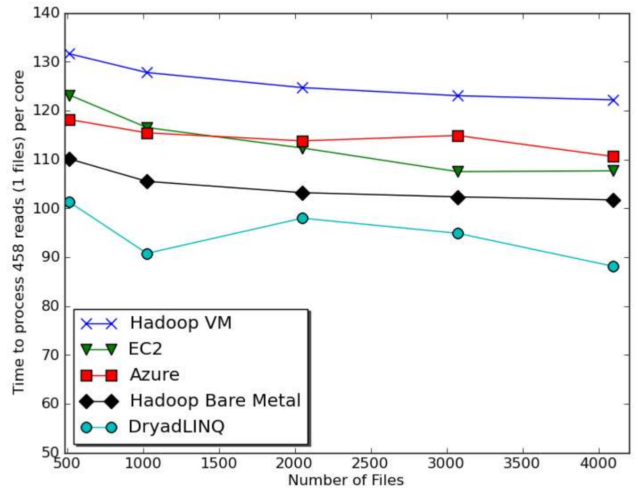

Figure 4 : Cap3 time to process a single file (458 reads) per core with different frameworks

<u>Cost to process 4096 FASTA files (~1GB) on EC2 (58 minutes)</u>

Compute 1 hour X 16 HCXL instances	*= 0.68$ * 16 = 10.88 $*
10000 SQS requests	*= 0.01 $*
Storage per 1GB per month	*= 0.15 $*
Data transfer out per 1 GB	*= 0.15 $*
Total	*= 11.28 $*

<u>Cost to process 4096 FASTA files (~1GB) on Azure (59 minutes)</u>

Compute 1 hour X 128 small instances	*= 0.12 $ * 128 = 15.36 $*
10000 SQS requests	*= 0.01 $*
Storage per 1GB per month	*= 0.15 $*
Data transfer in/out per 1 GB	*= 0.10 $ + 0.15 $*
Total	*= 15.77 $*

In addition to the above cost, there will be additional costs for the instance uptime for environment preparation and minor miscellaneous platform specific charges for things such as number of storage requests.

Based on figure 4 we can conclude that all four implementations exhibit reasonable efficiency within reasonable limits. When interpreting figure 5, it should be noted that the Cap3 program performs ~12.5% faster on windows environment than on the Linux environment. As we mentioned earlier we cannot use these results to claim that a given framework performs better than another, as only approximations are possible as the underlying infrastructure configurations of the cloud environments are unknown. Still, one interesting result to note in figure 5 is that the EC2 performance is better than the Hadoop VM performance. In the above performance tests the Cap3 Hadoop implementation relied on a shared file system for data storage rather than on the HDFS, which might have contributed to the lower performance of Hadoop Cap3 implementation. In the figure 6 & 7 we compare the performance of Cap3 Hadoop application using HDFS and using a shared file system on a 32 node X 24 core cluster. It clearly shows the bottleneck of the shared file system and network I/O. The difference is much bigger in the 24 maps per node case as the concurrent load on the network and file system is much higher.

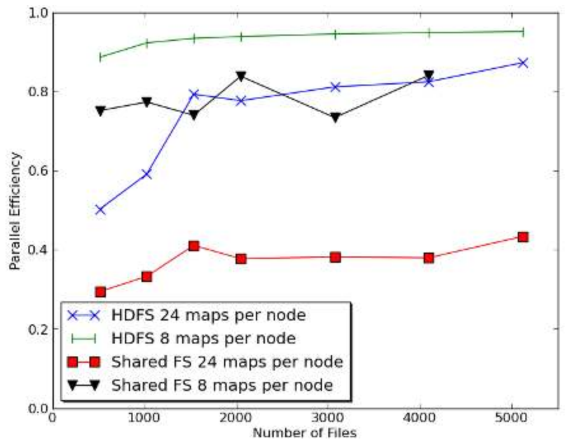

Figure 5 : Hadoop Cap3 parallel efficiency using shared file system vs HDFS on a 768 core (24 core X 32 nodes) cluster

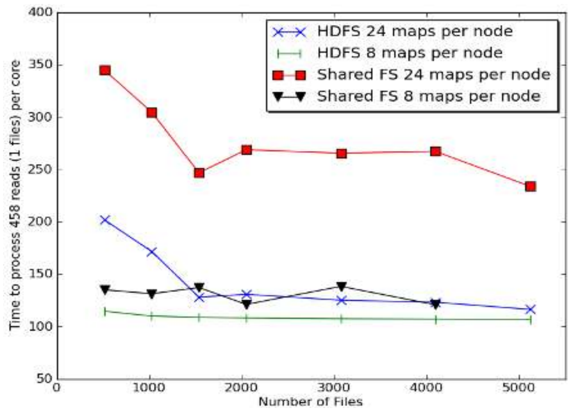

Figure 6 : Hadoop Cap3 performance shared FS vs HDFS

4.2.2 GTM Interpolation

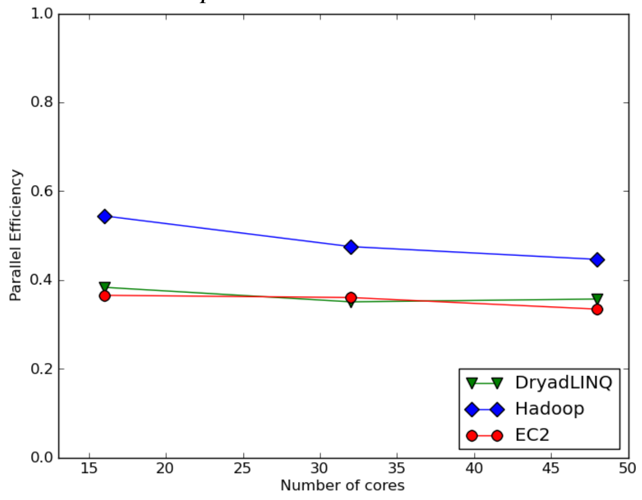

Figure 7: GTM interpolation efficiency on 26 Million Pubchem data points

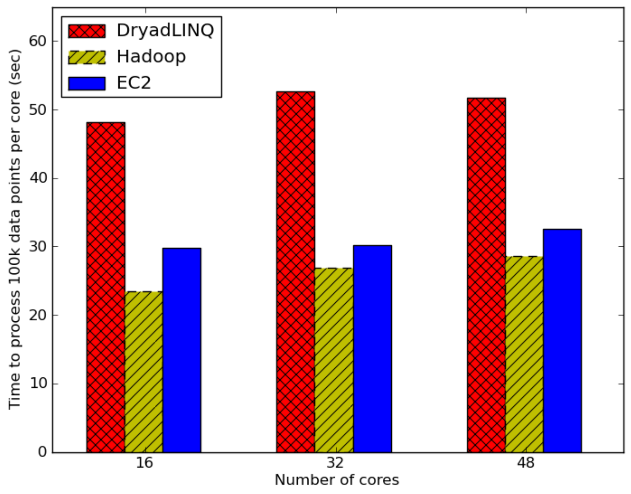

Figure 8 : GTM interpolation performance on 26 Million Pubchem data set

We used the pubchem data set of 26 million data points with 166 dimensions to analyze the GTM interpolation implementations. We used a 100,000 already processed subset of the data as a seed for the GTM interpolation. EC2 Cap3 tests were performed on EC2 High CPU Extra Large instances (considered as 8 cores) as they give the best cost efficiency according to the table 3. DryadLINQ Cap3 tests were performed on a 16 core (AMD Opteron 2.3 Ghz) per node, 16GB memory per node cluster. Hadoop Cap3 tests were performed on a 24 core (Intel Xeon 2.4 Ghz) per node, 48 GB memory per node cluster which was configured to use only 8 cores per node.

Once again we can notice that the 3 frameworks perform comparably. Characteristics of the GTM interpolation application are different from the Cap3 application as GTM is more memory intensive and the memory bandwidth becomes the bottleneck, which gives rise to the lower efficiency numbers. Also the grain size of the computational tasks in the GTM application was relatively smaller than Cap3 or MDS interpolation. Compressed data splits, which were unzipped before handing over to the executable, were used due to the large size of the input data. When the input data size is larger, Hadoop & DryadLINQ applications have an advantage of data locality based scheduling over EC2. Hadoop & DryadLINQ model brings computation to the data optimizing the I/O load, while the classic cloud model brings data to the computations.

4.2.3 MDS interpolation

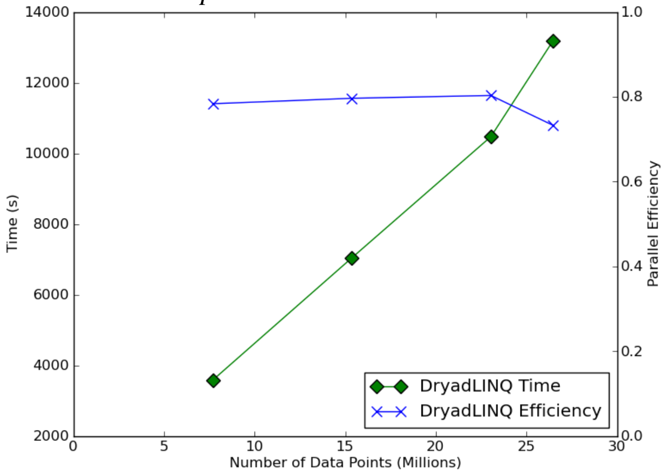

Figure 9 : DryadLINQ MDS interpolation performance on a 768 core cluster (32 node X 24 cores)

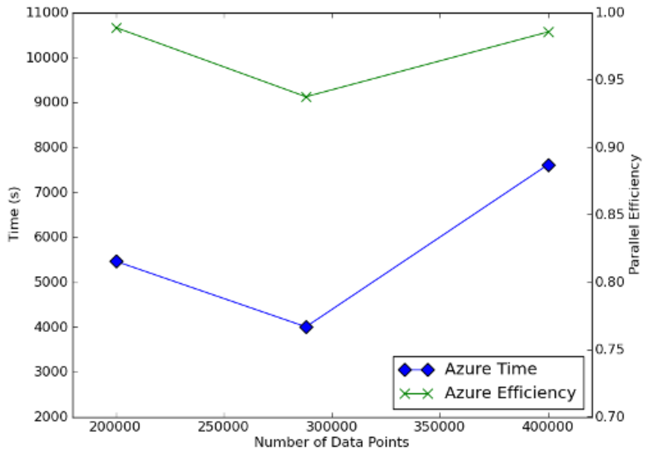

Figure 10 : Azure MDS interpolation performance on 24 small Azure instances

Figure 9 presents the results for processing millions of data points, broken in parts of 10000 data points, from the Pubchem data set with the DryadLINQ MDS interpolation application using 100,000 already MDSed seed data. A 24 core (Intel Xeon 2.4 Ghz) 48 GB memory per node 32 nodes cluster was use for this study. The dip of efficiency we notice in the figure 9 from 23 million to 26 million is due to the unbalanced partition that occurs when distributing 2600 parts among 768 cores.

Figure 10 presents results for Azure MDS interpolation application using 24 Azure small instances using a much smaller data set. The Azure MDS interpolation application exhibits very good efficiency, especially when considering the input and output data transfers it needs to perform from and to the Azure blob storage. The efficiency difference between DryadLINQ MDS interpolation & Azure MDS interpolation should be due to the fact that Azure small instances have a single exclusive memory per core while in the DryadLINQ cluster 24 cores have to share a single memory bus.

5. RELATED WORKS

There exist many studies [16-18] of using existing traditional scientific applications and benchmarks on the cloud. In contrast in this paper we focused on implementing and analyzing the performance of biomedical applications using cloud services/technologies and cloud oriented programming frameworks.

In one of our earlier work[15] we analyzed the overhead of virtualization and the effect of inhomogeneous data on the cloud oriented programming frameworks. Also Ekanayake and Fox[18] analyzed the overhead of MPI running on virtual machines under different VM configurations and using different MPI stacks.

CloudMapReduce[19] is an effort to implement a map reduce framework using the cloud infrastructure services. Amazon AWS[3] also offers Map Reduce as a cloud service through Elastic Map Reduce.

In addition to the biomedical applications we have discussed in this paper, we also developed distributed pair-wise sequence alignment applications using the Map Reduce programming models[15]. There are also many other bio-medical applications developed using Map Reduce programming frameworks such as CloudBLAST[20], which implements BLAST algorithm and CloudBurst, which performs parallel genome read mappings.

6. CONCLUSION

We have demonstrated that clouds offer attractive computing paradigms for two loosely coupled scientific computation applications. Cloud infrastructure based models as well as the Map Reduce based frameworks offered very good parallel efficiencies given sufficiently coarser grain task decompositions. The higher level MapReduce paradigm offered a simpler programming model. Also by using two different kinds of applications we showed that selecting an instance type which suits your application can give significant time & monetary advantages. Our previous work has tackled a broader range of data intensive applications under MapReduce and also compared them to MPI on raw hardware. The cost effectiveness of cloud data centers combined with the comparable performance reported here suggests that loosely coupled science applications will increasingly be implemented on clouds and that using MapReduce will offer convenient user interfaces with little overhead.

7. REFERENCES

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

[2] X. Huang, & Madan, A., "CAP3: A DNA sequence assembly program.," Genome Res, vol. 9, pp. 868-77, 1999.

[3] Amazon Web Services. Available: http://aws.amazon.com/

[4] Windows Azure Platform. Available: http://www.microsoft.com/windowsazure/

[5] Apache Hadoop. Available: http://hadoop.apache.org/core/

[6] Yu Y, Fetterly D, Budiu M, Erlingsson U, Gunda PK, Currey J, "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language," presented at the Symposium on Operating System Design and Implementation (OSDI), San Diego, CA, 2008.

[7] M. Isard, Budiu, M., Yu, Y., Birrell, A., & Fetterly, D., "Dryad: Distributed data-parallel programs from sequential building blocks," presented at the ACM SIGOPS Operating Systems Review, 2007.

[8] Y. Yu, Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P., et al., "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. ," Symposium on Operating System Design and Implementation (OSDI), 2008.

[9] Kruskal JB, Multidimensional Scaling. Beverly Hills, CA, U.S.A: Sage Publications Inc, 1978.

[10] J. Y. Choi, "Deterministic Annealing for Generative Topographic Mapping GTM," September 2 2009.

[11] T. Kohonen, "The self-organizing map," Neurocomputing, vol. 21, pp. 1--6, 1998.

[12] C. M. Bishop and M. Svensén, "GTM: A principled alternative to the self-organizing map," Advances in neural information processing systems, pp. 354--360, 1997.

[13] A. Dempster, et al., "Maximum Likelihood from incomplete data via the EM algorithm," Journal of the Royal Statistical Society. Series B, pp. 1--38, 1977.

[14] Jong Youl Choi, Seung-Hee Bae, Xiaohong Qiu and Geoffrey Fox, "High Performance Dimension Reduction and Visualization for Large High-dimensional Data Analysis " Technical Report submitted for publication, November 14 2009.

[15] Jaliya Ekanayake, Thilina Gunarathne, Judy Qiu, "Cloud Technologies for Bioinformatics Applications," Indiana University2010.

[16] E. Walker. http://www.usenix.org/publications/login/2008-10/openpdfs/walker.pdf. ;login: The USENIX Magazine. Available: http://www.usenix.org/publications/login/2008-10/openpdfs/walker.pdf

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

[18] Jaliya Ekanayake, Geoffrey Fox, "High Performance Parallel Computing with Clouds and Cloud Technologies," presented at the First International Conference CloudComp on Cloud Computing, Munich, Germany, 2009.

[19] cloudmapreduce. Available: http://code.google.com/p/cloudmapreduce/

[20] A. T. Matsunaga, M. Fortes, J., "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications " in IEEE Fourth International

Conference on eScience (eScience '08), Indianapolis, IN 2008.