**Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications**

Author(s) Name(s)

*Author Affiliation(s)*

*E-mail*

**Abstract**

Cloud computing offers exciting new approaches for scientific computing that leverages the hardware and software investments on large scale data centers by major commercial players. Loosely coupled problems are very important in many scientific fields and are on the rise with the ongoing move towards data intensive computing. There exist several approaches to leverage clouds & cloud oriented data processing frameworks to perform pleasingly parallel computations. In this paper we present two pleasingly parallel biomedical applications, 1) assembly of genome fragments 2) dimension reduction in the analysis of chemical structures, implemented utilizing cloud infrastructure service based utility computing models of Amazon Web Services and Microsoft Windows Azure as well as utilizing MapReduce based data processing frameworks, Apache Hadoop and Microsoft DryadLINQ. We review and compare each of the frameworks and perform a comparative study among them based on performance, efficiency, cost and the usability. Cloud service based utility computing model and the managed parallelism (MapReduce) exhibited comparable performance and efficiencies for the applications we considered. We analyze the variations in cost between the different platform choices (eg: EC2 instance types), highlighting the need to select the appropriate platform based on the nature of the computation.

# 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 rates by the now widely available automated DNA Sequencer machines. Another example would be the data generated by the Large Hadron Collider. Preprocessing, processing and analyzing these large amounts of data is a unique very challenging problem, yet opens up many opportunities for computational as well as computer scientists. According to Jim Gray, increasingly the scientific breakthroughs will be powered by computing capabilities that support researchers to analyze massive data sets. He aptly named data intensive scientific discovery as the forth science paradigm of discovery[1].

Cloud computing offerings by major commercial players provide on demand computational services over the web, which can be purchased within a matter of minutes simply by use of a credit card. The utility computing model offered through those cloud computing offerings opens up exciting new opportunities for the computational scientists to perform their computations since such a model suits well for the occasional resource intensive staccato compute needs of the scientists. Another interesting feature for scientists is the ability to increase the throughput of their computations by horizontally scaling the compute resources without incurring additional cost overhead. For an example in a utility computing model, 100 hours of 10 compute nodes cost same as 10 hours in 100 compute nodes. This is facilitated by the virtually unlimited resource availability of cloud computing infrastructures backed by the world’s largest data centers owned by the major commercial players such as Amazon, Google & Microsoft. We expect the economies of scale enjoyed by the cloud providers scale would translate to cost efficiencies for the users.

In addition to the leasing of virtualized compute nodes, cloud computing platforms also 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 technologies and frameworks. One example would be the Map Reduce[2] framework, which allow users to effectively perform distributed computations in increasingly brittle environments such as commodity clusters and computational clouds. Apache Hadoop[3] and Microsoft DryadLINQ[4] are two such parallel data processing frameworks which supports Map Reduce type computations.

A pleasingly parallel (also called embarrassingly parallel) application is an application which can be parallelized requiring minimal effort to divide the application in to independent parallel parts, each of which have no or very minimal data, synchronization or ordering dependencies among each other. These applications are good candidates for commodity compute clusters with no specialized interconnects. There are many scientific applications that fall in to this category. Few examples of pleasingly parallel applications would be Monte Carlo simulations, BLAST searches, many image processing applications such as ray tracing, parametric studies. Most of the data cleansing and pre-processing applications can also be classified as pleasingly parallel applications. The relative number of pleasingly parallel scientific workloads has been growing recently due to the emerging data intensive computational fields such as bioinformatics.

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[5] sequence assembly, GTM interpolation and MDS 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. We use the GTM and MDS interpolation applications to perform dimension reduction on 166-dimensional dataset containing 26 million data points obtained from the PubChem project database. We use Amazon Web Services[6] and Microsoft Windows Azure[7] cloud computing platforms and use Apache Hadoop[3] Map Reduce and Microsoft DryaLINQ[4] as the distributed parallel computing frameworks. BLAST

# 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 cloud programming models and the application frameworks we developed using them to perform pleasingly parallel computations. These frameworks have been used to implement the applications mentioned in section 3.

## Classic cloud architecture

### Amazon Web Services. Amazon Web Services (AWS)[6] 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 allowing 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. EC2 follows an infrastructure as a service approach where it provides users with ‘root’ access to the virtual machines giving maximum possible flexibility. Users can store virtual machines snapshots as Amazon Machine Images (AMIs), which can be used as templates to create new instances. Amazon EC2 offers a variety of hourly billed instance sizes with different price points giving a richer set of options for the user to choose from depending on his requirements. 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. The clock speed of a single EC2 compute unit is approximately 1 GHz to 1.2 GHz. The small instance type with a single EC2 compute unit is only available in 32-bit x86 environment, while the larger instance types support 64 bit x86\_64 environment as well.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 8 X (~3.25Ghz) | 2.40$ |

 | Table 2 : Microsoft Windows Azure instance types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instance Type | CPU Cores | Memory | Local Disk Space | Cost per hour |
| Small | 1 | 1.7 GB | 250 GB | 0.12$ |
| Medium | 2 | 3.5 GB | 500 GB | 0.24$ |
| Large | 4 | 7 GB | 1000 GB | 0.48$ |
| Extra Large | 8 | 15 GB | 2000 GB | 0.96$ |

 |

 SQS 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. SQS provides a REST based web service interface enabling any HTTP capable client to use it. 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. SQS 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. The service is priced based on the number of API requests as well as based on the total amount of data transfer per month.

S3 provides a web-scale distributed storage service where users can store and retrieve any type of data through a web services interface. S3 is accessible from anywhere in the web. Data objects in S3 are access controllable and can be organized in to buckets. S3 pricing is based on the size of the stored data, amount of data transferred and the number of API requests.

### Microsoft Azure Platform. Microsoft Azure platform[7] is a cloud computing platform offering 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. Features of the Azure services are more or less similar to the features of the AWS services we discussed above, except for the following.

Windows Azure Compute only supports Windows virtual machine instances and offers a limited variety of instance types when compared with Amazon EC2. As shown in Table 2, Azure instance type configurations and the cost scales up linearly from small, medium, large to extra large. Azure instances are available in 64 bit x86\_64 environment. It’s been speculated that the clock speed of a single CPU core in Azure terminology is approximately 1.5 GHz to 1.7 GHz. During our performance testing using the Cap3 program (section 3.1), we found that 8 Azure small instances perform comparable to a single Amazon high CPU extra large instance with 20 EC2 compute units. This lead us to believe that one Azure small instance CPU core performance is equal to 2.5 EC2 compute units, though this might be different for larger Azure 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. Users do not have the ability to interact with the Azure instances, other than through the deployed programs.

### Classic cloud processing model

|  |  |
| --- | --- |
| C:\Users\thilina\Documents\papers\pubchem\classic_cloud_1.emfFigure 1: Classic cloud processing model | Figure 2: Hadoop MapReduce processing model |

Figure 1 depicts the architecture of the classic cloud processing model. Varia[8] and Chappell[9] describe similar architectures that are implemented using Amazon and Azure processing models respectively. The classic cloud processing model follows a task processing pipeline approach with independent workers. It uses the cloud instances (EC2/Azure Compute) for data processing and uses Amazon S3/Windows Azure Storage for the data storage. For the task scheduling pipeline, it uses an Amazon SQS or an Azure queue as a queue of tasks where every message in the queue describes a single task. The client populates the scheduling queue with tasks, while the worker processes running in cloud instances pick tasks from the scheduling queue. The configurable visibility timeout feature of SQS and Azure queue, where a message will not be visible to other workers for the given amount of time once a worker reads it and reappears after the timeout, is used to provide a simple fault tolerance capability to the system. The workers delete the task (message) in the queue only after the completion of the task. Hence, a task (message) will get processed by some worker if the task does not get completed with the initial reader (worker) within the given time limit. Multiple instances processing the same task or another worker re-executing a failed task will not affect the result due to the idempotent nature of the independent tasks.

For the applications discussed in this paper, a single task comprises of a single input file and a single output file. The worker processes will retrieve the input files from the cloud storage through the web service interface using HTTP 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 to process the tasks 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. One interesting feature of the classic cloud framework is the ability to extend it to use the local machines and clusters side by side with the clouds. Even though it might not be the best option due to the data being stored in the cloud, one can start workers in computers outside of the cloud to help perform the computations.

## Apache Hadoop MapReduce

Apache Hadoop[3] is an open source implementation of the Google MapReduce[2] technology and shares many characteristics with the Google MapReduce implementation. Apache Hadoop MapReduce uses HDFS 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 commodity clusters and achieves reliability through replication of file data. When executing Map Reduce programs the data is typically stored in HDFS and Hadoop optimizes the data communication by scheduling computations near the data using the data locality information provided by the HDFS file system. Hadoop follows a master node with many client workers approach and uses a global queue for the task scheduling, achieving natural load balancing among the tasks. Hadoop performs data distribution and automatic task partitioning based on the information provided in the master program and based the structure of the data stored in HDFS. The Map Reduce model reduces the data transfer overheads by overlapping data communication with computation when reduce steps are involved. Hadoop performs duplicate execution of slower tasks and handles failures by rerunning of the failed tasks using different workers.

As shown in figure 2, the pleasingly parallel application framework on Hadoop is developed as a set of map tasks which process the given data splits (files) using the configured executable program. Input to the Hadoop map task comprises of key, value pairs, where by default Hadoop parses the contents of the data split to read them. Most of the time the legacy 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.

### Amazon Elastic MapReduce.Amazon Elastic MapReduce is an Amazon Web Service which offers a hosted Hadoop framework using Amazon EC2 for computation and Amazon S3 for input and output data storage. Elastic MapReduce allows the users to perform Hadoop MapReduce computations in the cloud through a web application interface as well as through a command line API without worrying about installing and configuring a Hadoop cluster. Hadoop MapReduce applications can be run on Elastic MapReduce with minimal or no changes. Elastic MapReduce provides logging and monitoring support utilizing the Amazon SimpleDB service. Version 0.18.3 is the Hadoop version supported by Elastic MapReduce (as of April 2010), while the current main stream Hadoop version is 0.20.2. There exist API differences between the two versions, making it cumbersome to backport the applications written for the latest Hadoop version to run on Elastic MapReduce.

## DryadLINQ

Dryad[10] 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. DAGs can be used to represent MapReduce type computations and can be extended to represent many other parallel abstractions too. Similar to the Map Reduce frameworks, the 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[4] 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 only on Window HPC clusters.

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.

## 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 and DryadLINQ. Hadoop and DryadLINQ take 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.

Table 3: Summary of cloud technology features

|  |  |  |  |
| --- | --- | --- | --- |
|  | AWS/ Azure | Hadoop | DryadLINQ |
| Programming patterns | Independent job execution, More structure can be imposed using client side driver program. | Map Reduce | DAG execution, Extensible to MapReduce and other patterns |
| Fault Tolerance | Task re-execution based on a time out  | Re-execution of failed and slow tasks. | Re-execution of failed and slow tasks. |
| Data Storage & Communication  | S3/Azure Storage. Data retrieved through HTTP. | HDFS parallel file system. TCP/IP based Communication layer | Local files  |
| Environment | EC2/Azure virtual instances, local compute resources | Linux cluster, Amazon Elastic MapReduce | Windows HPCS cluster |
| Ease of Programming | EC2 : \*\*Azure: \*\*\* | \*\*\*\* | \*\*\*\* |
| Ease of use | EC2 : \*\*\* Azure: \*\* | \*\*\* | \*\*\*\* |
| Scheduling & Load Balancing | Dynamic scheduling through a global queue, providing natural load balancing | Data locality, rack aware dynamic task scheduling through a global queue, providing natural load balancing | Data locality, network topology aware scheduling. Static task partitions at the node level, suboptimal load balancing |

EC2 and Azure classic cloud implementations involved more effort than the Hadoop and 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 SDK provides better development and testing support through the visual studio integration. The local development compute fabric and the local development storage of the Azure SDK make it much easier to test and debug the Azure applications. Azure platform is heading towards providing a more developer friendly environment. But as of today (Mar 2010) the Azure platform is less matured compared to the AWS, with deployment glitches and with the non-deterministic times taken for the deployment process.

# Performance Methodology

Perf on instance types as well as total

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. The horizontal axis labeling is in the format ‘Instance Type’ – ‘Number of Instances’ X ‘Number of Workers per Instance’. For an example, HCXL – 2 X 8 means two HighCPU-ExtraLarge instances are used with 8 workers per instance.

## Performance comparison between different implementations

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

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

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 different technologies due to several reasons. Even though efficiency accounts the system differences which affect 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 affect 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.

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

# Cap3

Cap3 [5] is a sequence assembly program which assembles DNA sequences by aligning and merging sequence fragments to construct whole genome 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 genomes at once. The Cap3 algorithm operates on a 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. The run time of the Cap3 application depends on the contents of the input file. Cap3 is relatively not memory intensive compared to the interpolation algorithms we discuss below. 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.

## Performance with different EC2 cloud instance types

Figure 3 present the benchmark results for Cap3 application on different EC2 instance types.

According to these results we can infer that memory is not a bottleneck for the Cap3 program and that the performance mainly depends on the computational power. While HighMemory-Quadruple ExtraLarge instances show the best performance due to the higher clock rated processors, the most cost effective performance for the Cap3 EC2 application is gained using the high CPU extra large instances.

|  |  |
| --- | --- |
| Figure 3 : Cap3 performance with different instance types to process 200 FASTA files with 458 reads in each file | Figure 4: EC2 GTM performance with different instance types to process 100,000 PubChem data points |

## Scalability results

We benchmarked the Cap3 classic cloud implementation performance using a replicated set of FASTA formatted data files, each file containing 458 reads, and compared with our previous performance results[17] for Cap3 DryadLINQ on bare metal, Cap3 Hadoop bare metal and Cap3 Hadoop on virtual machine . 16 High CPU extra large instances were used for the EC2 testing and 128 small Azure 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 calculations. In all the cases it is assumed that the data was already present in the frameworks preferred storage location.

|  |  |
| --- | --- |
| cap3_EC2_Hadoop_Dryad_eff.pngFigure 5 : Cap3 parallel efficiency | fig4.pngFigure 6 : Cap3 time to process a single file (458 reads) per core with different frameworks |

Load balancing across the different sub tasks do not pose a significant overhead in the Cap3 performance studies, as we used a replicated set of input data files making each sub task identical. We performed a detailed study of the performance of Hadoop and DryadLINQ in the face of inhomogeneous data in one of our previous works[17], where we noticed better natural load balancing in Hadoop due to its dynamic global level scheduling than in DryadLINQ, which uses static task partitioning. We assume cloud frameworks will be able perform better load balancing similar to Hadoop as they share the same dynamic scheduling global queue architecture.

## Cost comparison

Below we estimate the cost to assemble 4096 FASTA files using classic computing implementations of EC2 and on Azure. There will be additional costs for the instance time required for environment preparation and minor miscellaneous platform specific charges such as number of storage requests. For the sake of cost comparison, we also approximate the cost for the computation using one of our internal compute clusters (32 node 24 core, 48 GB memory per node with Infiniband interconnects) , with the cluster purchase cost depreciated over 3 years in addition to the yearly maintenance fee, which includes power and administration costs.

Cost to process 4096 FASTA files (~1GB) on EC2 (58 minutes)

Compute 1 hour X 16 HCXL instances = 0.68$ \* 16 = 10.88 $

10000 SQS messages = 0.01 $

Storage per 1GB per month = 0.15 $

Data transfer out per 1 GB = 0.15 $

Total = 11.19 $

Cost to process 4096 FASTA files (~1GB) on Azure (59 minutes)

Compute 1 hour X 128 small instances = 0.12 $ \* 128 = 15.36 $

10000 Queue messages = 0.01 $

Storage per 1GB per month = 0.15 $

Data transfer in/out per 1 GB = 0.10 $ + 0.15 $

Total = 15.77 $

Cost to process 4096 FASTA files (~1GB) on the local cluster (10.9 minutes) using Hadoop

Purchase price ≈ 500,000 $

Yearly maintenance fee ≈ 150,000 $

Cost for computation assuming 60% utilization ≈ 11.01 $

Cost for computation assuming 70% utilization ≈ 9.43 $

Cost for computation assuming 80% utilization ≈ 8.25 $

Based on figure 5 & 6 we can conclude that all four implementations exhibit similar (within 20%) reasonable efficiency with low parallelization overheads. When interpreting figure 6, 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 given that the underlying infrastructure configurations of the cloud environments are unknown. Still, one interesting result to note in figure 6 is that the EC2 performance being better than the Hadoop virtual machine performance. Unfortunately we were not able to determine the Cap3 sequential processing time on the Hadoop virtual machine environment to calculate the parallel efficiency values for the Hadoop VM tests. In the above Hadoop & Hadoop VM performance tests, the Cap3 Hadoop implementation relied on a shared network file system for data storage rather than on the HDFS, which might have contributed to the lower performance of the Hadoop Cap3 implementation

# BLAST

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

For BLAST+ execution, the application requires a few important parameters, such as a query and database. The database is normally large, with a size up to several gigabytes. Although BLAST+ is much faster than original BLAST (written in C) with multi-thread support, 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. Even though replicated, unlike in Cap3, the data set is inhomogenous.

Each Input query file contained 100 queries with a size of 7-8 KB. Size of output files ranged from 0 to 1 MB..

## Instance Types

## Scalability

|  |  |
| --- | --- |
|  |  |

BLAST : distribution : inhomogeneous unlike Cap3 -> draw graph of individual times.

Inhomogeneity in the data set.. Second wave interlaced with the first wave.

## Cost comparison

# GTM and MDS Interpolation

Multidimensional Scaling (MDS)[11] and Generative Topographic Mapping **(**GTM)[12] are dimension reduction algorithms which finds an optimal user-defined low-dimensional representation out of the data in high-dimensional space. These dimension reduction algorithms play a key role in scientific data visualization. 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 while 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 an objective function.

**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) [13]which does not have any density model, GTM defines an explicit density model based on Gaussian distribution [14] and finds the best set of parameters associated with Gaussian mixtures by using an Expectation-Maximization (EM) optimization algorithm[15].

Interpolation of MDS[16] and GTM are out-of-sample extensions of the original algorithms and they are 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 the 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.

 The size of the input data for the interpolation algorithms consisting of millions of data points usually ranges in gigabytes, while the size of the output data in lower dimensions is orders of magnitude smaller than the input data. The input data can be partitioned arbitrarily on the data point boundaries to generate computational sub tasks. The output data from the sub tasks can be collected using a simple merging operation and does not require any special combining functions. The GTM interpolation application is high memory intensive and requires large amount of memory proportional to the size of the input data. The MDS interpolation is less memory bound compared to GTM interpolation, but is much more CPU intensive than the GTM interpolation.

jks

## Application performance with different cloud instance types

According to the figure 4, 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 number of processor cores sharing the memory.

The high memory quadruple extra large instances give the best performance, but still the high CPU extra large instances appear as the most economical choice.

### GTM Interpolation

We used the PubChem data set of 26.4 million data points with 166 dimensions to analyze the GTM interpolation applications. PubChem is a NIH funded repository of over 60 million chemical molecules including their chemical structures and biological activities. We used a 100,000 already processed subset of the data as a seed for the GTM interpolation. We partitioned the input data in to 264 files with each file containing 100,000 data points. Figure 8 and 9 depicts the performance of the GTM interpolation implementations.

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. Classic cloud Azure tests we performed on Azure small instances where a single instance is considered as a single core in the figure 10. Classic cloud EC2 tests were performed on EC2 large, High CPU Extra Large (HCXL) as well as on High Memory Quadruple Extra Large (HM4XL) instances separately. HM4XL and HCXL instances were considered 8 cores per instance while ‘Large’ instances were considered 2 cores per instance.

|  |  |
| --- | --- |
| fig9.pngFigure 9: GTM interpolation efficiency on 26 Million PubChem data points | fig10.pngFigure 10 : GTM interpolation performance on 26.4 Million PubChem data set |

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 we assume as the cause of the lower efficiency numbers. Among the EC2 different instances, large instances achieved the best parallel efficiency and HM4XL instances gave the best performance while HCXL instances were the most economical. Azure small instances achieved the overall best efficiency. The efficiency numbers highlight the memory bound nature of the GTM interpolation computation, where platforms with less memory contention (less CPU cores sharing a single memory) performed better. As we can notice, the DryadLINQ GTM interpolation efficiency is lower than the others. One reason for the lower efficiency would be the usage of 16 core machines for the computation, which puts more contention on the memory.

GTM application computational tasks were much finer grain than in 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 and DryadLINQ model brings computation to the data optimizing the I/O load, while the classic cloud model brings data to the computations.

### MDS interpolation

|  |  |
| --- | --- |
| mds.pngFigure 11 : DryadLINQ MDS interpolation performance on a 768 core cluster (32 node X 24 cores)  | fig10.pngFigure 12 : Azure MDS interpolation performance on 24 small Azure instances |

Figure 11 presents the results for processing millions of data points, broken in parts of 10000 data points, from the PubChem data set using the DryadLINQ MDS interpolation application. 100,000 already MDSed seed data was used for the interpolation performance tests. A 24 core (Intel Xeon 2.4 Ghz) 48 GB memory per node 32 nodes cluster was used for this study. The plunge of efficiency we notice in the figure 11 from 23 million to 26 million is due to the unbalanced partition that occurs when distributing 2600 parts among 768 cores.

Figure 12 presents results for the 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 and 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. It was not possible to perform MDS interpolation using Hadoop and EC2 classic cloud framework as MDS interpolation application required .net framework.

# RELATED WORKS

There exist many studies [18-20] 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 works[17] we analyzed the overhead of virtualization and the effect of inhomogeneous data on the cloud oriented programming frameworks. Also Ekanayake and Fox[21] analyzed the overhead of MPI running on virtual machines under different VM configurations and using different MPI stacks.

CloudMapReduce[21] is an effort to implement a map reduce framework using the cloud infrastructure services. Amazon AWS[6] also offers Map Reduce as a cloud service through Elastic Map Reduce. AzureMapReduce

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[17]. There are other bio-medical applications developed using Map Reduce programming frameworks such as CloudBLAST[22], which implements BLAST algorithm and CloudBurst[23], which performs parallel genome read mappings.

Cloud HPC performance papers

# CONCLUSION & FUTURE WORK

We have demonstrated that clouds offer attractive computing paradigms for three loosely coupled scientific computation applications. Cloud infrastructure based models as well as the Map Reduce based frameworks offered 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 and 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 frameworks will offer convenient user interfaces with little overhead.