Development and Evaluation of Cloud Computing Infrastructures for Next-Generation Sequence Data Analysis

by Vivekananda Sarangi

BTech in Bioinformatics, May 2010, D.Y Patil University, India

A Thesis submitted to

The Faculty of
The Columbian College of Arts and Sciences
of The George Washington University
in partial fulfillment of the requirements
for the degree of Master of Science

August 31, 2012

Thesis directed by

Konstantinos Krampis
Assistant Professor of Informatics, J. Craig Venter Institute

Acknowledgement

I am heartily thankful to my supervisor, Dr Konstantinos Krampis, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. His guidance helped me in all time of research, experiments and writing of this thesis. I could not have imagined having a better advisor and mentor for my thesis project. It was a really good learning experience working under him.

My sincere thanks go to Dr Raja Mazumder for his precious advice throughout my Master's program at the George Washington University and helping me to take the right decisions at the right time.

I would like to thank Dr Jack Vanderhoek for making my journey through the Master's program so smooth and fruitful.

I would also like to my friends and colleagues at The George Washington University and J. Craig Venter institute for making the working environment filled with positivity and for their support.

Last but not the least I would like to thank my family for raising me and supporting me in every aspect of my life. All my credit goes to them.

Abstract

Development and Evaluation of Cloud Computing Infrastructures
for Next-Generation Sequence Data Analysis

Background

After the completion of the Human Genome Project, there has been a high demand for low cost sequencing. This has given rise to many high throughput Next Generation Sequencing techniques. Following this there has been a steep increase in the amount of sequencing data. This has created some problems for the biologists which include storage of the data, processing of the data and sharing the data. While large genomic institutes like Broad Institute and J Craig Venter Institute have the necessary infrastructure and resource to process these data, the same becomes very difficult for a small laboratory with individual researcher. With the advent of bench top genome sequencers like Miseq from Illumina and GS Junior from Roche, small labs can generate huge amount of data from complete genome sequencing of viral, bacterial and fungal genome in very less time. With this data small labs need additional fund for building clusters, for hiring experts to manage the clusters. With the increase in data they will have to upgrade the infrastructure. It also leads to minimal utilization of the hardware and duplication of data across labs. Cloud computing can be a viable solution to this problem. Researchers can rent computation capacity on demand from companies like Amazon and Google AppEngine which rent computational resource in a pay as you go model. This way they don't have to invest on buying and maintaining any hardware, and they don't have to pay when not using it. In order to make the cloud infrastructure more compatible with biological workflows certain approaches has been made. These approaches include development of Cloud BioLinux which offers an on-demand, cloud computing solution for the bioinformatics community, and is available for use on private or publicly accessible, commercially hosted cloud computing infrastructure such as Amazon EC2 and CloVR which is a desktop application for push-button automated sequence analysis that can utilize cloud computing resources to provide improved access to bioinformatics workflows and distributed computing resource. In this study we use the metagenomic pipeline from the J Craig Venter private cluster to study its behavior on the Amazon EC2 cloud using Cloud BioLinux virtual machine.

Result

The metagenomic pipeline was executed on the Amazon EC2 using the Cloud Biolinux virtual machine with different configurations. First a preliminary experiment was conducted using only one input file. Based on the results of this test the final experiment was designed by making changes to the preliminary experiment. The final experiment was conducted with four input files. Attributes were selected from the usage over the entire cluster and the values were stored in Excel worksheets. From these worksheets, graphs were produces showing the relation of the attributes with the addition of nodes to the cluster. Based on these graphs the behavior of the pipeline was studied and discussed.

Conclusion

Based on the result of the execution of the pipeline on Amazon EC2 with different configuration the behavior of the pipeline was studied. All the attributes that were tested, only the relevant attributes which affected the efficiency and behavior of the pipeline were discussed. Among these attributes we have some attributes whose values increases as the number of nodes increased, whereas the expected result was a decrease in the value with the increase in nodes. For these attributes it was concluded that, when a small size of data is processed over a smaller size cluster, the overheads and the network latency contribute to the increase in the value. For some attributes the value increased when a bigger master is used as compared to the smaller master. Some attributes are constant across the nodes. Any spike in these attributes can be suggested to be an abnormality in the execution of the pipeline. There were also some attributes which were dependent on the sequences specific to the input file. Using the values of these attributes we can understand the behavior of the pipeline and make changes to it in order to make it more efficient inside the cloud infrastructure.

Table of Content

Acknowledgement ii

Abstract iii

List of Figures vi

1. Introduction 1

1.1 Definition of 'Cloud Computing' 1
1.2 Next Generation Sequencing and Cloud Computing 6
1.3 Metagenomics 11

2. Methods 12

2.1 The pipeline 12
2.2 Moving the pipeline onto Amazon EC2 15
2.3 Experiment - 1 16
2.4 Experiment - 2 17

3. Result 21

4. Discussion 25

5. References 26

6. Figures 31

List of Figures

FIGURE-1: Shows S3 pricing from http://aws.amazon.com/s3/pricing/ 31
FIGURE-2: Eucalyptus infrastructure. 31
FIGURE-3: Galaxy cloudman interface. 32
FIGURE-4: Preliminary test Configurations. 32
FIGURE-5: Final configurations 33
FIGURE-6: Preliminary Tests 33
FIGURE -7: CPU versus virtual nodes 33
FIGURE-8: MEM versus virtual nodes 34
FIGURE-9: Wallclock time versus nodes 34
FIGURE-10: Maximum Virtual memory versus virtual nodes 35
FIGURE-11: Minor page fault versus virtual nodes with HMM 35
FIGURE- 12: Minor page fault versus virtual nodes with only BLAST 35

1. Introduction

James Watson and Francis Crick working together in the University of Cambridge, England discovered the structure of the DNA double helix in 1953[1]. Twenty five years after the discovery, in 1977 the first complete genome was sequenced (bacteriophage φX174)[2]. In the same year Allan Maxam and Walter Gilbert published a method for DNA sequencing by chemical degradation [3] and Frederick Sanger published a method for DNA sequencing with chain-terminating inhibitors independently [4]. Since then scientists have been trying hard to find more suitable and cost effective way to sequence DNA. The completion of 'The Human Genome Project' in 2001 [5][6] futher accerlerated the research for finding the best and cheapest sequencing technique. Ever since the 'Human Genome Project' there has been a steep drop in the cost of the next-generation sequencing. The benchtop genome sequencers like MiSeq from Illumina [7] has made complete sequencing of viral, bacterial and fungal genome affordable for small laboratories. It provides 'push button' ease and delivers results in hours. This allows the small laboratories to generate a large amount of data in little time. But to process the data these labs need large scale computational capacity, which demands for investment in computer hardware and skilled bioinformaticians. With the increase in the data generated, the required computational infrastructure will outspace what these labs will be able to support. But this problem can be dealt with by a new computational model known as 'Cloud Computing'.

1.1 Definition of 'Cloud Computing'

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and software systems in the datacenters that provide those services.When a Cloud is made available in a pay-as-you-go manner to the public, it is called a Public Cloud (a cluster that offers virtualized computational resource) [8]. Current examples of Public Clouds include AmazonWeb Services [9], Google AppEngine [10] and Microsoft Azure [11]. The term Private Cloud refers to internal virtualized datacenters of a business or other organization that are not made available to the public, such as Eucalyptus [12], OpenNebula [13] and Nimbus [14].

From a hardware point of view, three aspects are new in Cloud Computing [15]:

1. The availability of a large pool of computing resources available on demand,

thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning infrastructure;

2. The elimination of an up-front capital investment by Cloud users, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs and;
3. The ability to pay for use of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them as needed, thereby conserving resources by releasing machines and storage go when they are no longer useful.

Amazon EC2 (Elastic Compute Cloud) is an Infrastucture-as-a- Service (IaaS) provided by Amazon with a 'pay as you go' usage based pricing . IaaS is a systems that give users the ability to run and control entire virtual machine instances deployed across a variety physical resources. It is the most popular and has become the standard for IaaS service providers. Amazon uses preconfigured operating systems inside AMIs (Amazons Machine Images)[9]. An AMI serves as the basic unit of deployment for computational services delivered using EC2. A virtual machine (VM) is a software implementation of a computer that executes programs like a physical machine. The actual physical computational resources are on a distant server, which has a virtualization layer on top of it. This virtualization layer creates virtual machine with its own virtual operating system, CPU and memory as per the users specification. These AMIs are used to create virtual machines within the Amazon EC2 through a virtualization application software. Each running instance of the virtual machine is called as a virtual machine instance or just an Instance. The available Instance operating system are Windows Server2003, Red Hat Linux, Fedora Core, openSUSE, Gentoo, Oracle Enterprise Linux, Ubuntu, and Debian Linux. Currently there are several other commercial clouds that offer these features, such as GoGrid [17] and FlexiScale [18]. It is also possible to build private clouds instead of using public clouds with open-source cloud computing middleware such as Eucalyptus[12], OpenNebula[13] and Nimbus[14]. Though EC2 cannot compete with a high-performance dedicated cluster with Myrinet or Infiniband interconnects (communications link used in high-performance computing and enterprise data centers), but it can be comparable with the commodity cluster of a scientific lab. The network latency of EC2 was also found to be higher than the local clusters. But when

combined with low cost and on demand accessibility it may provide an alternative to dedicated clusters [19]. Thus Amazon EC2 can be imensely helpful to small laboratories which doesn't have access to large high performance computing.

Once the small laboratories start using the table top sequencers and analyze/assemble the data using EC2, a large amount of data will be generated. Storing this data again will need large storage capacity. Apart from the data generated by the small laboratories other organizations performing large scale physics experiments such as DZero [20], LHC[21], or SLAC [22] generate more than 1 terabyte of data daily [23]. Managing such huge data results in high storage and management cost. Amazon provides a storage service called as S3 (Simple Storage Service). It provides storage at a low cost, high availability service with a 'pay as you go' model. FIGURE-1 shows the prices offered by S3. The scientific data stored in S3 can be cheaply processed using virtual EC2 machines. Data stored in S3 is organized over a two-level namespace. At the top level are buckets similar to folders or containers. They have a unique global name. They serve several purposes: they allow users to organize their data; they identify the user to be charged for storage and data transfers. Each Amazon Web Services account may have up to 100 S3 buckets. Each bucket can store an unlimited number of data objects. Each object has a name, an opaque blob of data and metadata consisting of a small set of predefined entries and up to 4KB of user-specified name/value pairs. Search is limited to queries based on object's name and to a single bucket. The user is assigned an identity key and a private key when they register for the Amazon's Web Services, using which one can access the S3 account. The security provided by S3 is dependent on these identity key and the private key. S3 supports three data access protocols namely SOAP (Simple Object Access Protocol: is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks) [24], REST (Representational State Transfer: emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency) [25], BitTorrent [26]. Using BitTorrent S3 can provide tracker and seed functionality which can save bandwidth when multiple concurrent clients are demanding the same set of objects. S3 has attracted a large user base due to its simple charging scheme, unlimited storage capacity, open protocols, and simple API for easy integration with applications. But the current S3 design needs to be improved before it

can provide durable storage for scientific community[27]. A storage infrastructure which aims at data intensive scientific community must provide data durability, data availability, access performance, usability, support for security and privacy and low cost. S3 show 100 percent durability, for concurrent performance and remote access performance with high competency but the cost of storing data for experiments like DZero[20] can cost upto $1.02 million for a year [27]. Storage cost can be reduced by storing 'cold' data (rarely used data) on low cost storage and maintaining only the data most likely to be used on high-availability, low-latency storage. Another way is to only store raw data and derive the rest from the raw data. This is called the Reduced Redundancy Storage which is a storage option in the Amazon S3 [28]. It allows users to cut down on the costs by storing non-critical, reproducible data at lower levels of redundancy than Amazon S3's standard storage

S3 doesn't provide any checkpoint or back up facility for data which are accidentally erased or modified. S3 charging scheme (doesn't support any user specified usage limit) also cannot avoid or handle a monetary loss if an attacker repeatedly transfers data to and from an S3 bucket or in case of a buggy program. These risks can be avoided by using S3 as the back end of a storage system, and having users connect to a front-end running on Amazon's EC2 service. The front-end would be responsible for individual account management, fine-grained trust decisions, and billing.[27]

Another way of dealing with the concern of high cost and security is to use private clouds like Eucalyptus. Eucalyptus is an open source software framework for cloud computing that implements what is commonly referred to as Infrastructure as a Service (IaaS). It emulates the Amazon interface. The flexibility of Eucalyptus is that it is made up of several components that interact with each other through well defined interfaces, and provides the option to replace the built in implementation with their own modules or modify the existing ones. Eucalyptus comprises of four high level components each with its own Web-service interface. The Node Controller, Cluster Controller, Storage Controller (Walrus) and Cloud Controller (FIGURE-2). The Node Controller (NC) executes on every node and is designated for hosting Virtual Machine instances. It controls the execution, inspection and termination of Virtual Machine instances on the host where it runs. It receives queries and control from its Cluster controller and in turn queries and controls the system software on its nodes accordingly. The queries are made to find out the node's physical resources like the

number of cores, memory size, available disk space as well as to learn about the state of the VM instances on the node. This information is propagated to the Cluster Controller in response to *describeResource* and *describeInstances* (this request lists out the number of instance that are created and whether they are running or pending). The Cluster Controller controls VM instances on a node by executing *runInstance* (when an instance is created this command is used run the instance) and *terminateInstance* (this requests is used to terminate a virtual machine instance) requests to the node's NC. To start an instance, the NC makes a node-local copy of the instance image files (the kernel, the root file system, and the ramdisk image), either from a remote image repository or from the local cache. It then creates a new endpoint in the virtual network overlay (so that is behaves like a physical machine on the network), and instructs the hypervisor (desktop virtualization software which moves the operating system and the applications it supports from the physical local desktop to a virtual computer running in a server/cloud. [29]) to boot the instance. To stop the instance, the NC instructs the hypervisor to terminate the VM. It then tears down the virtual network endpoint, and cleans up the files associated with the instance. The Cluster Controllers (CC) generally executes on a cluster front-end machine which has network connectivity to both the nodes running the NCs and the machine on which the Cloud Controller is running. It has three primary functions: it schedules incoming run requests to specific Node Controllers, controls the instance virtual network overlay and gathers or reports information about a set of NCs.

The Storage Controller is a data storage service called Walrus. It is interface compatible with Amazon S3. Walrus implements REST as well as SOAP interface that is compatible with S3. Users who have access to Eucalyptus can use Walrus to stream data in/out of the cloud. Users can use standard S3 tools like s3cmd (It is a command line client for copying files to/from Amazon S3 and performing other related tasks, for instance creating and removing buckets, listing objects [30]) to stream data in and out of Walrus. To aid scalability,Walrus does not provide locking for object writes. But as with S3, users are guaranteed that a consistent copy of the object will be saved if there are concurrent writes to the same object. If a write to an object is encountered while there is a previous write to the same object in progress, the previous write is invalidated. Walrus also acts as an VM image storage and management service. The Node Controller sends an image download request to

Walrus before instantiating it on a node. After authenticating the request by using an internal set of credentials, the images are verified, decrypted and finally transferred.

Cloud Controller is the entry point of the user and administrator into the cloud. It queries node manager for information, make high level scheduling decisions and implements them by making requests to cluster controllers. It is a collection of web services, based on there roles they are grouped into three categories. First, the Resource Services allows users to manipulate properties of the virtual machines and networks, and monitor both system components and virtual resources. Second, Data Services govern persistent user and system data and provide for a configurable user environment for formulating resource allocation request properties. Lastly, Interface Services present user-visible interfaces, handling authentication & protocol translation, and expose system management tools providing. Eucalyptus has made cloud computing design space flexible by providing a system that is easy to implement on top of existing resources. Making it open source and modular has exposed it to experimentation and provides powerful features out-of-the-box through an interface compatible with Amazon EC2 [31].

1.2 Next Generation Sequencing and Cloud Computing

Next Generation Sequencing (NGS) refers to technologies that do not rely on traditional dideoxy-nucleotide (Sanger) sequencing where labeled DNA fragments are physically resolved by electrophoresis. These new technologies rely on different strategies, but essentially all of them make use of real-time data collection of a nucleotide base level incorporation event across a massive number of reactions (on the order of millions versus 96 for capillary electrophoresis for instance)[32]. Next generation sequencing technologies can be distinguished from Sanger sequencing in that they do not use chain termination chemistry and electrophoresis. Instead they rely on the amplification of single DNA molecules to generate clusters of DNA templates held at defined locations on a solid support. This procedure is called solid phase amplification. These clusters of identical molecules are then sequenced in parallel by cyclic incorporation and measurement of fluorescently labelled nucleotides (Illumina) or short oligonucleotides (ABI SOLiD), or the detection of by products (454/Roche). Because of the parallel sequencing of the amplified clusters, this technology is also called massive parallel sequencing or high throughput sequencing. The first NGS platform became commercially available in 2005, the FLX Genome

Sequencer by 454 Life Sciences [33], now owned by the F. Hoffmann LaRoche group (Switzerland). In January 2007 Illumina Inc. (San Diego, California, USA) launched their Genome Analyzer I [34] and the Applied Biosystems(ABI) SOLiD [35] became available in October of the same year. The major commercial Next Generation Sequencing platforms available to researchers are the 454 Genome Sequencer (Roche) [36] which was the first commercial platform introduced in 2004 as the 454 Sequencer, Illumina (formerly Solexa). Genome analyzer [37] was the second platform to reach market and currently is the most widely used system. The SOLiD system (Applied Biosystems/Life Technologies)[38] is another platform that uses a unique sequencing-by ligation approach in which it uses an emulsion PCR approach with small magnetic beads to amplify the DNA fragments for parallel sequencing [39] and the Heliscope (Helicos Corporation) [40]. The Helicos HeliScope platform is the first single molecular sequencing technology available that uses a highly sensitive fluorescence detection system to directly detect each nucleotide as it is synthesized [39].

Sequence throughput from these new generation of instruments continues to increase exponentially at the same time that the cost of sequencing a genome continues to fall [41]. The introduction of bench top genome sequencers such as MiSeq from Illumina have made it possible for small labs to acquire the technology. Storage and management of the data generated is arguably the largest issue with which a small lab facility will struggle as a mere 10–20 sequencing runs (Illumina) could overwhelm any storage and archiving system available to an individual investigators. A 36 cycle run from MiSeq (sequencing 36 base pairs) takes about four hours to complete and generates an output greater than 1 GB. So for 20 runs a huge data of 20 GB is produced in a matter of four hours. With every 150 cycle run an output of 3 GB is generated resulting in 60 GB for 20 runs [42]. At this rate the data generated in a few days or weeks will exhaust the storage infrastructure of a small lab. Furthermore, discovery with the data requires large scale computational analysis. For this the laboratories has to invest substantially in computer hardware and skilled informatics. An alternate to building and maintaining local hardware (clusters) is to use 'Cloud computing'.

Once such effort is the 'Cloud Biolinux' project [47]. It offers an on-demand, cloud computing solution for the bioinformatics community, and is available for use on

private or publicly accessible, commercially hosted cloud computing infrastructure such as Amazon EC2. Cloud BioLinux is publicly available through the Amazon EC2 cloud [9]. The Cloud BioLinux VM can also be executed on an open-source Eucalyptus cloud, or directly on a desktop computer using virtualization software like Virtualbox [43]. The Cloud BioLinux project aims to provide a configurable, automated framework for building VMs with biological software to small laboratories without access to large computational resources.

Cloud Biolinux is build on the bioinformatics packages, documentation and desktop interface of NEBC BioLinux release 6.0 [44]. NEBC BioLinux contains 137 bioinformatics packages, including the blastall and blast+ NCBI applications, the Staden, EMBOSS, hmmer, and phylip collections of software, many stand-alone applications for tasks such as sequence alignment, clustering, assembly, display, editing, and phylogeny, as well as tools for working with next generation sequencing data.

Virtual machines have a provision for whole system snap shot. Cloud BioLinux take advantage of this property of VMs to encapsulate the software tool, the operating system and the data within into a single digital image [45]. These images are ideal for reproducibility of in-silico analysis. This image contains all the changes that were made to the VM from the time it was started till the creation of the snapshot. This mechanism can be ideal for data sharing between collaborators. After completion of a particular computation the researcher can create a whole system snapshot and make the data or software accessible to collaborators by granting them access to the snapshot on Amazon EC2. For this the collaborators need to have accounts on this cloud platform. The other way is to make the snapshot available for download and execution on private clouds.

Cloud BioLinux can be accessed in the following simple steps through a web browser. First, the user has to create an Amazon EC2 account and login to the cloud console. After that from within the console he can use the “Launch Instance” button for launching Cloud BioLinux. After that he has to specify which Cloud BioLinux VM image he wants to launch. Then he has to select the computational capacity of the instance and give a password for remote desktop login. After doing this an internet address to the VM for the launched Cloud BioLinux is provided. The user has to copy this address to the remote desktop client and using the specified password he can

establish a connection after which he has access the full desktop session. The script and the configuration files are freely available from the GitHub code repository [46], for software developers to download and edit the configuration file the bioinformatics tool to be included in the Cloud BioLinux. As oppose to traditional clusters a users receive distinct and pre-defined computational resources, it prevents over-utilization of the hardware by a single user. Hadoop/MapReduce being already a part of Cloud BioLinux, the next version of VM plans to implement specialized scripts that allow end-users to easily provision Hadoop clusters on any cloud. This will facilitate running large-scale bioinformatics data processing pipelines [47].

CloudMan from the Galaxy project [48] was developed as an integrated solution that leverages existing tools and packages by providing a generic method for utilizing those tools on cloud resources and abstracting out low-level informatics details. It takes all the bioinformatic tools available in the NERC BioLinux [43] work station and provides a generic method for utilizing it on the cloud resources. All interaction with Galaxy CloudMan and its associated cloud cluster management is performed through a web based user interface which requires no computational expertise. The application currently supports the creation of a compute cluster on Amazon EC2 infrastructure. Galaxy CloudMan is ideal for small labs setting with independent researchers who have specific or periodic need for computational resource without having to worry about maintaining a computer cluster. There are three steps for instantiating a CloudMan compute cluster: First one has to create an Amazon Web Services account, then use the AWS Management Console to start a master EC2 instance, third use the CloudMan web console on the master instance to manage the cluster size. Once this is set up additional users can use the cluster through Galaxy web interface. CloudMan on EC2 uses Amazon's Elastic Block Storage (EBS) volumes for data storage. The size of the cloud cluster can be changed during run time by adding or removing worker nodes through the CloudMan web interface. The CloudMan web interface is also used to terminate all services and worker nodes when a given cluster in no longer needed. This can be done so by terminating the master node, this will automatically terminate all the worker nodes. The source code for the entire project is available under the MIT license and is available from http://bitbucket.org/galaxy/cloudman/. The domain specific tools are separated from the core components that enable CloudMan to operate. This enables the developers to

update the functionality of CloudMan without requiring the users to alter their routine. Thus it allows the user to focus on using the tools while the CloudMan developers can focus on ensuring the infrastructure works properly [48].

CloVR (Cloud Virtual Resource)[49], is a new desktop application for push-button automated sequence analysis that can utilize cloud computing resources to provide improved access to bioinformatics workflows and distributed computing resource. It provides a single VM containing pre-configured and automated pipelines, suitable for easy installation on the desktop and with cloud support for increased analysis throughput. CloVR adressess some of the technical issues with the cloud computing infrastructure: 1. it simplifies use of cloud computing platforms by automatically provisioning resources during pipeline execution, 2. it use local disk for storage and avoiding reliance on network file systems, 3. it provides a portable machine image that executes on both a personal computer and multiple cloud computing platforms. A shared folder has to be created to access data stored in the local computer. This shared folder should be accessible from both the VM and the local computer and can use the available hard disk space on the local computer. Once in the shared folder CloVR can use this data for processing, similarly its writes the output to the same shared folder. The CloVR can also be configured to automatically access a cloud provider for additional resources. The clouds supported by CloVR include the commercial Amazon Elastic Compute Cloud, the academic platforms DIAG [50] and Magellan [51]. Multiple copies of CloVR run simultaneously and interact as a cluster for parallel processing of data. Latest CloVR image as well as several reference datasets are maintained permanently in Amazon S3 for future reference. On the cloud, clusters of CloVR VM instances are configured for parallel processing. The client CloVR VM running on a local computer manages all communication and data transfer between the host's computer and the cloud. CloVR uses local disks and does not rely on network file systems during pipeline execution. The CloVR VM (version 0.6) contains four prepackaged automatic analysis tool. They are, a parallelized BLAST search protocol, a comparative 16S rRNA sequence analysis pipeline, a comparative metagenomic sequence analysis pipeline and a single microbial genome assembly and annotation pipeline. For these protocols some of the supported input file formats are SFF, FASTA, QUAL and FASTQ. The outputs are in standard formats such as FASTA and GENBANK flat file format including summary reports, tables and graphical

representaion of the analysis result[49]. Among the tools incorporated in CloVR is Cunninghum. It is a tool which estimates BLAST runtimes for shotgun sequence datasets using sequence composition statistics. It estimates the BLAST runtime from a given set of sequences, a BLAST program (BLAST{N,P,X}) and a pre specified database [52].

1.3 Metagenomics

More than 99 % of microbes present in the environment are not culturable in a lab [53]. This leads to the lack of access of these microorganism for biotechnology and basic research. These microbes can't be studied unless a method for their cultivation is found out. A large number of microbes present in the environmental sample cannot be cultured using standard culturing techniques, due to which the study of the genetic diversity present in the environment becomes limited. The task of finding different methods for cultivation for these millions of microbes is a seemingly impossible task. To overcome these difficulties several other methods were devised. The term "metagenomics" was first used in 1998 [54] to describe the study of collective genetic material from all microbes from a community sample. The collective genetic pool of microrganism in a community sample is called the metagenome and its study is called metagenomics. As metagenomics involve the cloning of genetic material directly from the environmental sample it bypasses the need for culturing the microbes hence saving a lot of time and cost. It also allows the study of the microbes in there natural habitat and is not biased toward culturable organism hence the entire genetic diversity of the enviromental sample and the interaction between organisms can be studied. Traditionally metagenomic studies were done by shotgun sequencing [55]. It involve various step in which the DNA is recovered from the environmental sample directly and is sheared into fragments. It is then cloned into vectors and transformed into suitable vectors to produce metagenomic libraries containing the inserts of the environmental DNA. While this process is effective in characterizing the microbial diversity in the sample, the cloning process is both laborious and costly. Thus next generation sequencing technologies opens the possibility to analyze the microbial community through direct sequencing without the initial cloning step. This technology is a fast high through put technique for sequencing DNA and is more suitable for metogenomic sequencing than the conventional Sanger sequencing [56].

As this technique is not biased to any microbial group and doesn't rely on known sequence information the chances of finding new species is also highly likely.

But sequencing of environmental sample have its own difficulties. In case of single organism genomics practically all the microorganism's genome is sequenced giving a clear idea of the genome. It is easier to know from which species the DNA came from. The location of genes, operons, and transcriptional units can be inferred computationally. But in case of metagenomes, we get a cocktail of DNA sequence from different species. For most of these species a full genome is not available and it is very difficult to determine the origin of the species. Depending on the sequencing methods used, the read lengths can be from 20 base pair to 700 base pairs. Short sequence reads which are dissociated from their original species can be assembled to a length not more than 5000 base pairs, thus construction of the whole genome is not possible [57]. One of the major problems of metagenomic sequences obtained from next generation sequencers is proccessing of short reads. Assembly of the short reads becomes difficult due to low coverage (ratio of length of the output and actual length of the sequence, more the coverage better the assembly). Due to the presence of a mixture of DNA sequence from different organism chimeric contigs are produced. And the absence of a reference genome makes mapping assembly also difficult.

2. Methods

To overcome the assembly problems of metagenomic sequence from next generation sequencer a metagenomic pipeline was developed at the J. Craig Venter Institute. The objective of the pipeline is to circumvent the assembly step and to find out the types of protein present in the environmental sample. This pipeline was used as a model for developing a cloud infrastructure for pipelines dealing with next generation sequence data analysis and performing performance matrices.

2.1 The pipeline

This pipeline is the pilot experiment of a project which aims to bypass the assembly of the reads to find out the protein content in the environmental sample. It tries to figure out if the protein composition/activity can be figured out by the reads of environmental sample from any next generation sequencer. The current thesis

concentrates on the infrastructure development to fit the cloud infrastructure, thus only a brief outline of what the pipeline does is described below. The pipeline takes a whole genome that is already fully assembled and its full-length proteins fully identified (.SEED files were obtained from TIGRFAMS [58] and the Pfam [59] libraries). It then cuts each protein sequence into fragments and each fragment is entered as input to the pipeline (with the extension .afa - aligned FASTA files). These chopped sequences are called as 'minis'. Each .SEED file can produce 10 – 20 minis with .afa extention. Each of the sequences in the .afa files are of 20 – 25 base pair long in order to mimic the reads from the Next Generation Sequencers.

Example:

The .SEED file is

```
# STOCKHOLM 1.0

OMNI|NTL01EC00023/1-87
LANIKSAKKRAIQSEKARKHNASRRSMMRTFIKKVYAAIEAGDK.......AAAQKAFNEMQPIVDRQAAKGLIHKNKAARHKANL
TAQINKLA
PIR|A64163|A64163/1-61
.........................MMRTYIKKVYAQVAAGEK.......SAAEAAFVEMQKVVDRMASKGLIHANKAANHKSKL
AAQIKKLA
OMNI|NTL01BS2548/1-87
MPNIKSAIKRTKTNNERGVHNATIKSAMRTAIKQVEASVANNEA.......DKAKTALTEAAKRIDKAVKTGLVHKNTAARYKSRL
AKKVNGLS
SP|P73336|RS20_SYNY3/1-94
MANIKSALKRIEIAERNRLQNKSYKSAIKTLMKKTFQSVEAYASDPNPEKLDTINTSMAAAFSKIDKAVKCKVIHKNNAARKKARL
AKALQSAL
OMNI|BB0233/23-107
LRKNASALKRSRQNLKRKIRNVSVKSELKTIEKRCINMIKAGKK.......DEAIEFFKFVAKKLDTAARKRIIHKNKAARKKSRL
NVLLLK..
SP|P75237|RS20_MYCPN/1-80
MANIKSNEKRLRQNIKRNLNNKGQKTKLKTNVKNFHKEINLDNL.......G......N.VYSQADRLARKGIISTNRARRLKSRN
VAVLNKTQ
SP|P55750|RS20_MYCGE/1-80
MANIKSNEKRLRQDIKRNLNNKGQKTKLKTNVKKFNKEINLDNL.......S......S.VYSQADRLARKGIISLNRAKRLKSKN
AVILHKSN
SP|P56027|RS20_HELPY/1-87
MANHKSAEKRIRQTIKRTERNRFYKTKIKNIIKAVREAVAVNDV.......AKAQERLKIANKELHKFVSKGILKKNTASRKVSRL
NASVKKIA
//
```

Then the mini files are as follows:

Mini.01.afa

```
>OMNI|NTL01EC00023/1-22
LANIKSAKKRAIQSEKARKHNA
>OMNI|NTL01BS2548/1-22
MPNIKSAIKRTKTNNERGVHNA
>SP|P73336|RS20_SYNY3/1-22
MANIKSALKRIEIAERNRLQNK
>OMNI|BB0233/23-44
LRKNASALKRSRQNLKRKIRNV
>SP|P75237|RS20_MYCPN/1-22
MANIKSNEKRLRQNIKRNLNNK
>SP|P55750|RS20_MYCGE/1-22
MANIKSNEKRLRQDIKRNLNNK
>SP|P56027|RS20_HELPY/1-22
MANHKSAEKRIRQTIKRTERNR
```

Mini.02.afa

```
>OMNI|NTL01EC00023/5-26
KSAKKRAIQSEKARKHNASRRS
>OMNI|NTL01BS2548/5-26
KSAIKRTKTNNERGVHNATIKS
>SP|P73336|RS20_SYNY3/5-26
KSALKRIEIAERNRLQNKSYKS
>OMNI|BB0233/27-48
ASALKRSRQNLKRKIRNVSVKS
>SP|P75237|RS20_MYCPN/5-26
KSNEKRLRQNIKRNLNNKGQKT
>SP|P55750|RS20_MYCGE/5-26
KSNEKRLRQDIKRNLNNKGQKT
>SP|P56027|RS20_HELPY/5-26
KSAEKRIRQTIKRTERNRFYKT
```

The pipeline uses HMMER 3 [60]. Its general usage is to identify homologous protein sequences by comparing a profile-HMM (Hidden Markov Models) to either a single sequence or a database of sequences. Sequences that score significantly better to the profile-HMM compared to a null model are considered to be homologous to the sequences that were used to construct the profile-HMM. Profile-HMMs are constructed from a multiple sequence alignment in the HMMER package using the hmmbuild program which reads a multiple sequence alignment file, builds a new profile HMM, and saves the HMM in a file. Normally in a HMMER search a .HMM file is created which is later used up by the program to give a list of top hits of protein to the query sequence.

The hmmbuild program read the .afa files and creates mini Hidden Markov Model (mHMM) storing it in a .HMM files. Profile HMM of the .SEED file is also created for calibration of the miniHMMs. The calibration is carried out by evaluating the hits to the parent model (i.e. profile HMM of the .SEED file) and each of its mHMMs against a test database. This is done by hmmsearch [61]. All the hits to the parent model which are above the trusted cutoff (a protein value above the cutoff value means the protein belong to the protein family) and span less than 85% of the parent model are considered as fragments and ignored. The hits to the mHMM are examined from its highest score to lower. The hits to the mHMM which are not in the set of sequences found in the trusted non-fragment hits group of the model may be considered as potential fragments. This is further tested by doing a BLAST [62] search of the sequences versus the test database. The hits which have scores above the trusted cutoff of the parent model are ignored. This also includes the hits to the strain of the same species. The hit list is processed in this way till the first non-fragment non-trusted hit to the parent model is identified. The score of this hit becomes the calibrated lower bound of the mHMM and the score of the lowest scoring true hit becomes the upper bound. The trusted cutoff is set at half the range of the upper and lower bound.

There are two processing intensive steps in this pipeline: one is the HMM search and the other is the BLAST search. The HMM is already parallelized using SGE (Sun Grid Engine) [64]. The BLAST search needs to be parallelized. It is done so by forking each of the BLAST process. The pipeline is written in Perl and the Perl Forks

module is used for parallelization. Detail about this process is explained in later part of this section.

2.2 Moving the pipeline onto Amazon EC2

In order to measure the performance matrix of the metagenomic pipeline on the cloud, one has to move the pipeline and the data set onto the cloud. To do so a Cloud BioLinux virtual machine instance was created on Amazon EC2 cloud computing infrastructure [9] with AMI id ami-********. The following command was used.

euca-run-instances –k gsc_key -t m1.xlarge -z us-east-1c ami-******** -f /home/vsarangi/.ama/gsc_cloudman_user_data -g CloudMan

euca-run-instance create an instance from the virtual machine image ami-********. –k command line option specifies the Amazon EC2 key which is gsc_key. –t specifies the memory size of the instance which in this case is 15 GB. –z specifies the zone (physical location of the server) in which the instance is created. It is important to keep the instance and the volume that we are going to attach below be in the same zone. –f specifies the user data file, it tell the instance to load any volume automatically according to the user specifications. –g specifies the security group. After this the command 'euca-describe-instances' is used to list all the running instances. To our instance (with ami-********) a 200 GB volume was attached by the following command.

euca-create-volume -s 200 -z us-east-1c

euca-attach-volume -i i-******** -d /dev/sdl vol-********

The –s in euca-create-volume specifies the storage size of the volume and –z specifies in which zone the volume need to be created. Care was taken to ensure that both the virtual machine instance and the volume were created on the same region / physical cluster so that the volume can be attached to the virtual machine. The –i in euca-attach-volume specifies the instance ID to which the volume needs to be attached followed by the instance ID. –d specifies the drive on the instance on which the volume will be mounted followed by the volume ID. The volume ID can be obtained after creation of the volume using the command 'euca-describe-volumes'. After

attaching the volume we need to log in to the instance to work in it. The secure shell (ssh) command is used to do so.

ssh -i gsc_key.pem ubuntu@ec2-50-17-133-69.compute-1.amazonaws.com

Where –i is for the key followed by the key. Ubuntu is the user and ec2-50-17-133-69.compute-1.amazonaws.com is the address of the instance on the cloud. Once we log into the instance we need to mount the volume that we created and attached using the command 'sudo mount /dev/xvdl /mnt'. The sudo stands for superuser do. It allows users to run programs with the security privileges of another user, normally the superuser, or root. 'dev/xvdl' is the drive onto which the volume will be mounted as 'mnt'. We can also unmount the volume by 'sudo umount /mnt'. The dataset and the pipeline were transferred to this volume from the local server at JCVI using:

scp -i gsc_key.pem /local/data ubuntu@ ec2-50-17-133-69.compute-1.amazonaws.com:/mnt1

Once inside the volume and the volume being attached to the instance the pipeline can now be run on the Amazon EC2. To learn how to start a Amazon EC2 readers can also refer to http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-gsg.pdf.

2.3 Experiment - 1

The first experiment is to modify the pipeline for parallelization of BLAST search as was mentioned earlier in the section. The pipeline does the HMM search using the profile HMM, mHMM against the test database. It uses SGE which is an open source batch queuing system developed and supported by Sun Microsystems, to send each of the computational jobs across the nodes of the cluster in parallel. Once this is done and we have the hits from the mHMMs we need to BLAST each of the qualified hits against the test database. The way the pipeline is written, it reads a single mHMM hit file, runs BLAST search on all its sequences in parallel using SGE and stores the result in a data structure. Though the BLAST search for all the sequences in a mHMM hit file are executed in parallel, all the other mHMM hit files wait till the completion of the previous file. Thus modifying the code to be able to execute each of the mHMM hit files in parallel will make the pipeline to execute faster. This was done so by using process forks. Instead of waiting for a mHMM hit file to complete its BLAST search all the mHMM hit files were send for BLAST search in parallel. The

Perl Forks library was used to copy the code for running BLAST with each mHMM hit file in parallel across different nodes. These forks were nothing but new processes doing the same thing with different files in parallel. The results of each of the fork were stored in the same data structure, so it was important to manage the access of the data structure among all the forks. A Perl module 'forks::shared' was used for that purpose. It can be downloaded on Linux platform using the command sudo perl –MCPAN –e 'install forks::shared'. It basically shares the address of the variable across the forks. The module also provides a way to detect deadlock and resolve it. Sometimes when two forks are trying to access the variable, a chance of deadlock is possible. The main program continues as the parent fork and the mHMM hit files continue as child forks. It is important for the parent fork to wait till all the child forks have finished their job. In the pipeline the data generated by the entire BLAST search (of all mHMM hit files) is necessary for the next steps, thus if the parent fork continues without the completion of all its children it won't have the necessary information to proceed and the program with terminate or hang. This was taken care of by keeping the parent fork busy in a loop which terminates only when execution of all the child forks were over. This was done by the SGE command 'qstat'. 'qstat' displays a list of the computational jobs that are being executed on the cluster along with their status and the node on which it is being executed. When the jobs are running on SGE 'qstat' always has a value and when all the jobs running on the cluster are over it returns an empty result. So a loop runs on the parent fork until the length of qstat is greater than zero. When the length of 'qstat' becomes zero the parent fork proceeds forward to remaining steps of the pipeline.

2.4 Experiment - 2

The second experiment is to find out how the pipeline behaves on the cloud and how we can optimize the amount of resources needed to run the pipeline efficiently. Once we have the optimized configuration we can incorporate the pipeline into Cloud BioLinux with the appropriate settings. To accomplish this we have to run the cluster in different configurations. This is done by using the Galaxy Cloudman framework, which supports the creation of a compute cluster on Amazon EC2. Galaxy Cloudman allows us to create one master node and many worker nodes. Using this setting we can study the behavior of the pipeline using different combination (the number of nodes used and/or the size of the nodes) of worker nodes and master nodes. Using AMI id

ami-******** a cloudman instance is created. We have to determine the size of the master node during the creation of the instance. Once the instance is created we can add as many worker nodes of various size (or same size) as we want. Galaxy cloud man provides an easy to use interface where one can add worker nodes by the click of a button. FIGURE 3 shows the interface of Galaxy cloudman at various stages.

To understand the behavior of the pipeline on the cloud a preliminary test was done in which four experiments were conducted on a single .SEED file based on which other tests were performed. In the first experiment a master node with 14.7 GB memory (4 virtual cores with 2 EC2 Compute Units each) was created and all the computational jobs were submitted to the cluster at once. The second experiment had the same setup but only four jobs were submitted at once. The third experimental setup had a master node of 68 GB memory (8 virtual cores with 3.25 EC2 Compute Units each) and it allowed all jobs to be submitted at once. The fourth experimental setup was same as the third only but in this case only four jobs were released at once. Each experimental setup was run with 2 worker nodes first followed by 4 nodes then 6 nodes and finally 8 nodes. Each worker nodes has 7.5 GB memory (2 virtual cores with 2 EC2 Compute Units each). The experimental details are shown in FIGURE 4. The jobs mentioned above are managed on the cloudman cluster using SGE. While the pipeline is running the jobs are submitted to the nodes if and when free. The jobs submitted by the pipeline are generated either by the HMM search and by the BLAST search by the program. The master node queues these jobs and schedules it to run on the Worker nodes if and when they are free. Each job submitted has a unique job name and job identification number (id). The details of each job can be extracted by using the SGE command ‘qacct’. The content of a typical ‘qacct’ output for the data of a job is shown below.

```
qname        all.q
hostname     ip-10-38-121-40.ec2.internal
group        ubuntu
owner        ubuntu
project      NONE
department   defaultdepartment
jobname      TIGR00009.HMM
jobnumber    106
taskid       undefined
account      sge
priority     0
qsub_time    Tue Jul 31 20:32:12 2012
```

```
start_time   Tue Jul 31 20:32:16 2012
end_time     Tue Jul 31 20:33:00 2012
granted_pe   NONE
slots        1
failed       0
exit_status  0
ru_wallclock 44
ru_utime     78.897
ru_stime     8.741
ru_maxrss    153280
ru_ixrss     0
ru_ismrss    0
ru_idrss     0
ru_isrss     0
ru_minflt    39625
ru_majflt    0
ru_nswap     0
ru_inblock   0
ru_oublock   4488
ru_msgsnd    0
ru_msgrcv    0
ru_nsignals  0
ru_nvcsw     22548
ru_nivcsw    111469
cpu          87.637
mem          30.801
io           4.197
iow          0.000
maxvmem      450.855M
arid         undefined
```

In this preliminary test we ran the pipeline with just one seed file and with pre-computed mHMM hit files (hits from the HMM search as explained in section 2.1). Thus the jobs that we got on running the pipeline in these four experiments are due to the BLAST search.

The job numbers were carefully noted before and after the execution of the pipeline as described in the experiments and the details of each job was extracted using ‘qacct -j’ followed by the job number. Each experiment gave four set of qacct outputs. These outputs were stored in four different Excel sheets. The rows in these Excel sheets store the different jobs and the columns store the values of the different parameters the qacct output provided. The same was done with all the experiments and we get 16 Excel sheets at the end of the test, four for each experiment. Using the values in these Excel sheets the performance of the pipeline were compared. For the preliminary test we took two attributes, execution time and system time. Using these comparisons we understood how the pipeline behaves in different configuration and it formed the basis

of further experiments. Each of these parameters was checked with the different number of worker nodes in each experiment to see how they behave when number of nodes increase or decrease. The following are the parameters that are provided by qacct and the parameters used for the preliminary test as well as further tests are in bold letters:-

qname - Name of the cluster queue in which the job has run
hostname – Name of the execution host
project – The project which was assigned to the job
department – The department which was assigned to the job
group – The effective group id of the job owner when executing the job
owner – Owner of the Grid Engine job
job_name – Name of the job
job_number – Job indentifier
task_number – Array job task index number
account – An account string as specified by the qsub.
priority – Priority value assigned to the job
submission_time – Submission time
start_time – Start time
end_time – End time
granted_pe – The parallel environment which was selected for the job
slots – the number of slots which were dispatched to the job by the scheduler
failed – Indicates the problem which occurred in case a job could not be started on the execution host
exit_status – Exit status of the job script
ru_wallclock – Difference between end_time and start_time, except that if the job fails, it is zero
ru_stime – User time used
ru_utime – System time used
ru_maxrss – Maximum resident set size
ru_ixrss – not used in Linux
ru_ismrss – not used in Linux
ru_idrss – Integral resident set size
ru_isrss - not used in Linux
ru_minflt – Page fault not requiring physical I/O
ru_majflt – Page fault requiring physical I/O
ru_nswap - Swaps
ru_inblock – Block input operations
ru_oublock – Blocks output operations
ru_msgsnd - not used in Linux
ru_msgrcv - not used in Linux
ru_nsignals - not used in Linux
ru_nvcsw - Voluntary context switches
ru_nivcsw – Involuntary context switches
cpu – The CPU time usage in seconds
mem – The integral memory usage in Gbytes seconds
io – The amount of data transferred in input/output operation (0 if unavailable)
iow – The input/output wait time in seconds (0 if unavailable)

maxvem – The maximum virtual memory size in bytes.
arid – Advance reservation identifier. If the job used the resource of on advance reservation, then this field contains a positive value otherwise its value is zero
pe_taskid – If this identifier is set, the task was part of a parallel job

The result of this comparison is mentioned in the section 3. From these experiments we realized that configuration 2 and configuration 4 were under utilizing the resources at hand and they were removed, thus further experiments were conducted using only two configurations. The new configurations are shown in FIGURE-5. The names of the configurations were changed completely to avoid confusion with the data in the preliminary tests. The config-1 is a smaller master node (14.7 GB) and config-2 is a bigger master node (68 GB).

This time the pipeline was run with four .SEED files individually. The pipeline was not run with pre-computed HMMs. Thus the jobs that we got on running the pipeline with these four files are due to the both the HMM and the BLAST search. The protocol for data and graph were the same as for the preliminary test only with different attributes and files. The attributes that were used are CPU time, memory, wallclock time, virtual memory, and page fault values. The details of these attributes and the comparison are presented in section 3.

3. Result

This section consists of two parts. The first part has one figure showing the result of the preliminary tests and the second part has figures of graphs that were generated from the final experiments.

FIGURE-6 shows the result of the preliminary tests consisting of the system time and the execution time. Execution time is the time taken by the pipeline in whole i.e from start till end executing all jobs. We get this time by just using the time command (time is a UNIX command which determines the duration of execution of a particular command, in our case it's the pipeline) in the command line before the execution of the pipeline. After the pipeline successfully completes it gives us the time. The graph in FIGURE-6a shows Config 1 takes the minimum time to execute, but the reduction in time over nodes is not significant. In case of Config 3, the execution time falls substantially with the increase in the number of nodes. It is almost close to the execution time of Config 1 with 8 nodes. Config 2 and 4 take more time to execute because by releasing 4 jobs at a time is an underutilization of the configuration. The

increase in the execution time is due to the waiting time of the cluster when the 4 jobs were being executed.

System time is the time the system uses to finish the job. The graph in FIGURE -6b shows the mean time taken for a single job to execute with respect to the increase in number of nodes. From the graph it is evident that with the increase in the virtual nodes the overhead increases over the master nodes resulting in the slow execution of the job. It is interesting to see that the System time of all the configs converges at a common point. This suggests that in this particular case increasing the size of the master nodes doesn't increase the efficiency of the pipeline. Comparing the two graphs we realize that Config 2 and Config 4 in which we are releasing only 4 forks at a time are underutilizing the resource, so it is preferred to continue the experiment without including them.

An experiment was developed as described in the Section 2, which involved four .SEED files and the configuration shown in FIGURE-5 to study the behavior of the pipeline in the Amazon EC2 cloud infrastructure. The four files that were taken were named TIGR00001.SEED, TIGR00009.SEED, TIGR00012.SEED, and TIGR00029.SEED. FIGURE -7 shows the CPU time from the qacct output of the jobs of all the files versus increasing the number of nodes. CPU time is the amount of time for which a central processing unit (CPU) was used for processing a computer program and in our case a process from a program. CPU time is the amount of time the CPU actual spends executing processor instructions and is often much less than the total program runs time. From FIGURE-7 we see that the values for CPU time for jobs versus increase in number of nodes is almost constant in all the four graphs (each representing a file). The constant value is due to the fact that the CPU processing cycles do not change with the addition or deletion of nodes as the computational tasks are identical (same sequences search against same databases). From the graph we can see that the value for Config 2 is smaller than that of Config 1. Config 1 uses a smaller master node because of which the overhead of managing the worker nodes is higher than that of Config 2 which has a bigger master node. This is the reason we see the change in CPU cycle in the two configurations. But these overheads add very little time to Config 1 due to which we see the two lines very close to each other. FIGURE-8 shows the graphs of the integral memory from the qacct output of the jobs of all the files versus the addition of nodes. The MEM field is the integral memory usage in Gbytes cpu second. It is basically the amount of memory used per CPU cycle. This

value also stays the same with the increase in the number of nodes. It is because the computational tasks remain the same (for a particular SEED file), as well as the CPU cycles requirement for completing the jobs remains the same. Hence the amount of memory used for the completion of the jobs for that particular file also remains the same. The explanation provided above for the CPU time also holds true for the integral memory.

FIGURE-9 shows the wallclock time from the qacct output of the jobs of all the files versus the increase in the number of nodes. This is the total time a program takes to execute, including the time it takes waiting for data transfer across the nodes or between a CPU and the disk or memory, in seconds. From the figure it is evident that there is an increase in the wallclock time with the increase in number of nodes. Ideally, when the number of nodes increases the wallclock time should reduce. As it has been demonstrated in previous studies [63] the increase in the wallclock time is because of network latency which increases with addition of nodes initially, but when the number of nodes is increased to over 30 the wallclock time falls considerable overcoming the network latency. Though the execution time of the pipeline drops with the addition of nodes, the wallclock time initially increases as the load to communicate between the nodes affects the load for processing of the jobs, while it decreases matching the same levels of the execution time as the number of studies gets increased. Unfortunately at this point of time, Galaxy Cloudman has been tested with only 20 nodes [48]; hence we could not test the behavior of wallclock time with more number of nodes.

FIGURE-10 shows the change in maximum virtual memory with the increase in the number of nodes. Linux divides its physical RAM (random access memory) into chucks of memory called pages. Swapping is the process where a page of memory is copied from the RAM to the hard disk, called swap space, to free up that page of memory. The combined size of the physical memory and the swap space is the amount of virtual memory available. When many processes are running in the memory (RAM) and all of a sudden one process requires more RAM than that is available, then that amount of memory is utilized from the swap space. Maxvem is the maximum virtual memory used by a process during its execution. Now in virtual memory, RAM is constant but the swap space increase with the requirement of the process (when a process demands more RAM than is available). Now in the graphs in FIGURE-10 we see that the Maxvem values of Config 2 with a bigger master is

higher than that of Config 1 with a smaller master. This can be explained by the fact that a bigger master can schedule more jobs than a small master. Hence with the same number of worker nodes, the RAM requirement of the nodes increase for Config 2 (bigger master), hence it takes more memory from the swap increasing maximum virtual memory. While in case of Config 1 for the same number and size of worker nodes, less number of jobs are released lowering the maximum virtual memory as compared to Config 2. It can also be observed that with the increase in the number of nodes the Maxvem starts to fall. In TIGR00001.SEED the Maxvem of both the configuration meet at the 8^{th} node. This suggests that with the increase in the number of nodes the RAM available to the number of jobs (which is constant for a SEED file) increase hence it requires less swap space and hence the value drops. The relation between Maxvem in Config 1 and Config 2 varies slightly in different file suggesting its SEED specific but the all tend to somehow converge to meet at a point. Hence if we increase the number of nodes the Maxvem value in all other SEEDs will eventually converge suggesting a point at which the size of the Master node will not be taken into account. If such a condition occurs then it will be more cost effective to rent a smaller Master over a bigger one.

As mentioned above Linux divides its physical RAM (random access memory) into chucks of memory called pages. An interrupt occurs when a program requests data that is not currently in real memory. This interrupt triggers the operating system to fetch the data from the virtual memory (swap on disk) and load it into the RAM. Page fault error occurs when the operating system cannot find the data in virtual memory. This usually happens when the virtual memory area, or the table that maps virtual addresses to real addresses, becomes corrupt. If the page is loaded in memory at the time the fault is generated, but is not marked in the memory management unit as being loaded in memory, then it is called a minor page fault. This could happen if the memory is shared by different programs and the page is already brought into memory for other programs. This minor page fault is recorded in the qacct for every job. FIGURE-11 shows the graphs of the minor page fault and the number of nodes. It shows the graph of two .SEED files: TIGR00029 and TIGR00009. The two graphs have the same pattern but one is the reverse of the other. In TIGR00009 Config 2 has a steep fall at the 6^{th} node and then it meets the Config 1 at the 8^{th} node. In TIGR00029 Config 1 has a steep fall at the 6^{th} node and meets Config 2 at the 8^{th} node. As explained earlier minor page fault happens when many processes are

running together and a page is loaded into memory without being registered to the memory management unit, so when other process call that memory it gives a page fault. Now we have two main processes in our pipeline which produces multiple process, one is the HMM and the other is the BLAST search. In order to find out which process produces the reversal of pattern in the two files, a graph was plotted only with the BLAST processes. The FIGURE-12 shows the graph showing minor page fault only for BLAST searches. And we see the same pattern arising again proving that the pattern is due to the BLAST search which is dependent on the SEED files. As we know that the seed files are created from the overlapping sequences. Thus after hmmsearch with the mHMM and the test database the resulting hit file also contains overlapping sequences. When one sequence is compared against the database the complete database has to be loaded into the memory for each BLAST and when an overlapping sequence hits the same database sequence a page fault is created. Hence we see the different patterns in the different files depending on the way the sequences are searched against the database in the memory.

4. Discussion

After the preliminary tests it was concluded that by releasing four forks at a time instead of all forks resulted in under utilization of the resources. Thus with the final test the configurations with four forks were omitted and the pipeline was tested with four .SEED files. Different attributes were compared between the files, namely CPU time, integral memory, wall clock time, maximum virtual memory and minor page fault value using the data from the qacct output. The details of these attributes are described in the result section. From the results it was found that some attributes values increases as the number of nodes increased, whereas the expected result was a decrease in the value with the increase in nodes. For these attributes it was concluded that, when a small size of data is processed over a smaller size cluster, the overheads and the network latency contribute to the increase in the value. Publications were cited explaining this abnormality [63]. Thus if we increase the number of nodes these values will fall considerable with the addition of new nodes. This has been seen in attributes like wall clock time and system time. Attributes like maximum virtual memory increased when a bigger master is used as compared to the smaller master. Though the value falls with the increase in number of nodes, it will be ideal to keep

the value low for better efficiency. As the efficiency of swap memory is less than that of the RAM, we need to release limited number of forks in order to keep its value low. Some attributes like CPU time and integral memory are constant across the nodes as expected. Any spike in these attributes can be suggested to be an abnormality. Values of attributes like minor fault page are dependent on sequences specific to the input files. It will be very difficult to do anything to change this value, but the study of this value can tell us about the behavior of a particular file with the pipeline. This can help us to modify other parameters to increase the efficiency of the pipeline. Using the values of these attributes we can understand the behavior of the pipeline and based on it make changes to make it more efficient inside the cloud infrastructure.

5. References

1. Watson J, Crick F. The structure of DNA. Cold Spring Harb Symp Quant Biol. 1953;18:123-31.

2. Sanger F, Air GM, Barrell BG, Brown NL, Coulson AR, Fiddes CA, Hutchison CA, Slocombe PM, Smith M. Nucleotide sequence of bacteriophage phi X174 DNA. Nature. 1977 Feb 24;265(5596):687-95.

3. Maxam AM, Gilbert W. A new method for sequencing DNA. Proc Natl Acad Sci U S A. 1977 Feb;74(2):560-4.

4. Sanger F, Nicklen S, Coulson AR. DNA sequencing with chain-terminating inhibitors. Proc Natl Acad Sci U S A. 1977 Dec;74(12):5463-7.

5. Venter JC, Adams MD, Myers EW. The sequence of the human genome. Science. 2001 Feb 16;291(5507):1304-51.

6. Lander ES, Linton LM, Birren B. Initial sequencing and analysis of the human genome. Nature. 2001 Feb 15;409(6822):860-921.

7. http://www.illumina.com/documents/products/datasheets/datasheet_miseq.pdf

8. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M: Above the Clouds: A Berkeley View of Cloud Computing. In Book Above the Clouds: A Berkeley View of Cloud Computing. University of California at Berkeley;Editor ed.eds. 2009:23.

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

10. Google AppEngine [http://code.google.com/appengine/]

11. Microsoft Azure [https://www.windowsazure.com/en-us/]

12. Eucalyptus [http://www.eucalyptus.com/]

13. "OpenNebula" [http://http://opennebula.org]

14. "Nimbus"[http://www.nimbusproject.org/]

15. Vogels W. A Head in the Clouds—The Power of Infrastructure as a Service. In First workshop on Cloud Computing and in Applications (CCA '08) (October 2008).

16. XEN [http://www.xen.org/]

17. GoGrid [http://www.gogrid.com/]

18. FlexiScale [http://www.flexiscale.com/]

19. Hill Z, Humphrey M. A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster?. 10th IEEE/ACM International Conference on Grid Computing, 2009.

20. The DZero Experiment. [http://www-d0.fnal.gov]

21. The Large Hadron Collider.[http://lcg.web.cern.ch/LCG]

22. The Stanford Linear Collider. [http://www2.slac.stanford.edu/vvc/experiments/slc.html]

23. The Compact Muon Solenoid at CERN. [http://cmsinfo.cern.ch]

24. http://www.w3.org/TR/soap/

25. Fielding, R. T. 2000. Architectural Styles and the Design of Network-Based Software Architectures, PhD Dissertation, University of California, Irvine, 2000

26. BitTorrent. [http://www.bittorrent.com]

27. Mayur Palankar, Adriana Iamnitchi, Matei Ripeanu, Simson Garfinkel. Amazon S3 for Science Grids: a Viable Solution?. DADC '08 Proceedings of the 2008 international workshop on Data-aware distributed computing.

28. http://aws.amazon.com/s3

29. "Virtualization in education". IBM. October 2007. Retrieved 6 July 2010

30. S3 Tools.[http://s3tools.org/s3cmd]

31. Daniel Nurmi, Rich Wolski, Chris Grzegorczyk Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. CCGRID '09 Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid.

32. Paul Richardson: Special Issue: Next Generation DNA Sequencing. Genes 2010, 1, 385-387; doi:10.3390/genes1030385

33. http://454.com

34. http://www.illumina.com

35. http://www3.appliedbiosystems.com

36. Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, et al. 2005. Genome sequencing in microfabricated high-density picolitre reactors. Nature 437:376–80

37. Bentley DR. 2006. Whole-genome resequencing. Curr. Opin. Genet. Dev. 16:545–52

38. http://www.appliedbiosystems.com/absite/us/en/home/applications-technologies/solid-next-generation-sequencing.html

39. Jun Zhang, Rod Chiodini, Ahmed Badr, and Genfa Zhang:The impact of next-generation sequencing on genomics. J Genet Genomics. 2011 March 20; 38(3): 95–109. doi:10.1016/j.jgg.2011.02.003.

40. www.helicosbio.com/Products/HelicosregGeneticAnalysisSystem/HeliScopetradeSequencer/tabid/87/Default.aspx

41. Brent G. Richter, David P. Sexton:Managing and Analyzing Next-Generation Sequence Data. June 2009 Issue of PLoS Computational Biology

42. http://support.illumina.com/sequencing/sequencing_instruments/miseq/questions.ilmn

43. Oracle VirtualBox [http://www.virtualbox.org]

44. Field D, Tiwari B, Booth T, Houten S, Swan D, Bertrand N, Thurston M: Open software for biologists: from famine to feast. Nature biotechnology 2006, 24:801–803

45. Dudley JT, Butte AJ: In silico research in the era of cloud computing. Nature Biotechnology 2010, 28:1181–1185.

46. Cloud BioLinux source code repository [https://github.com/chapmanb/cloudbiolinux]

47. Konstantinos Krampis, Tim Booth, Brad Chapman, Bela Tiwari, Mesude Bicak, Dawn Field and Karen E. Nelson. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. (2011) BMC Bioinformatics (in press).

48. Enis Afgan, Dannon Baker, Nate Coraor, Brad Chapman, Anton Nekrutenko, James Taylor. Galaxy CloudMan: delivering cloud compute clusters. BMC Bioinformatics 2010, 11(Suppl 12):S4

49. Samuel V Angiuoli1, Malcolm Matalka, Aaron Gussman, Kevin Galens, Mahesh Vangala, David R Riley, Cesar Arze, James R White, Owen White and W Florian Fricke. CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing. BMC Bioinformatics 2011, 12:356

50. DIAG. [http://diagcomputing.org/]

51. http://www.magellanglobalhealth.com/technologies/index.html

52. James Robert White, Malcolm Matalka, W. Florian Fricke & Samuel V. Angiuol. Cunningham: a BLAST Runtime Estimator.

53. Streit WR, Schmitz RA. Metagenomics--the key to the uncultured microbes. Curr Opin Microbiol. 2004 Oct;7(5):492-8.

54. Handelsman J, Rondon MR, Brady SF, Clardy J, Goodman RM. Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. Chem Biol. 1998 Oct;5(10):R245-9.

55. Sanger F, Nicklen S, Coulson AR. DNA sequencing with chain-terminating inhibitors. Proc Natl Acad Sci U S A. 1977 Dec;74(12):5463-7.

56. Erick Cardenas, James M M. Tiedje. New tools for discovering and characterizing microbial diversity. Current opinion in biotechnology, doi:10.1016/j.copbio.2008.10.010.

57. Wooley JC, Godzik A, Friedberg I. A primer on metagenomics. PLoS Comput

Biol. 2010 Feb 26;6(2):e1000667.

58. Daniel H. Haft,Jeremy D. Selengut, Owen White. The TIGRFAMs database of protein families. Nucl. Acids Res. (2003) 31 (1): 371-373. doi: 10.1093/nar/gkg128

59. PFAM:[http://pfam.sanger.ac.uk/]

60. HMMER:[hmmer.janelia.org/]

61. HMMSEARCH.[http://hmmer.janelia.org/search/hmmsearch]

62. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990 Oct 5;215(3):403-10.

63. C. L. Müller. Benchmark runs of pCMALib on Nehalem and Shanghai nodes. *MOSAIC technical report*, MOSAIC Group, ETH Zürich, April 2009.

64. SGE:[http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html]

6. Figures

	Standard Storage	Reduced Redundancy Storage
First 1 TB / month	$0.125 per GB	$0.093 per GB
Next 49 TB / month	$0.110 per GB	$0.083 per GB
Next 450 TB / month	$0.095 per GB	$0.073 per GB
Next 500 TB / month	$0.090 per GB	$0.063 per GB
Next 4000 TB / month	$0.080 per GB	$0.053 per GB
Over 5000 TB / month	$0.055 per GB	$0.037 per GB

FIGURE-1: Shows S3 pricing from http://aws.amazon.com/s3/pricing/

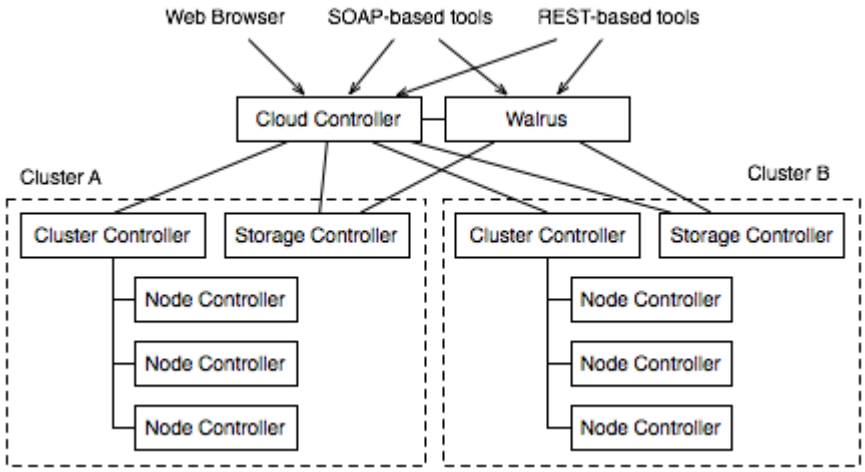

FIGURE-2: Eucalyptus infrastructure.

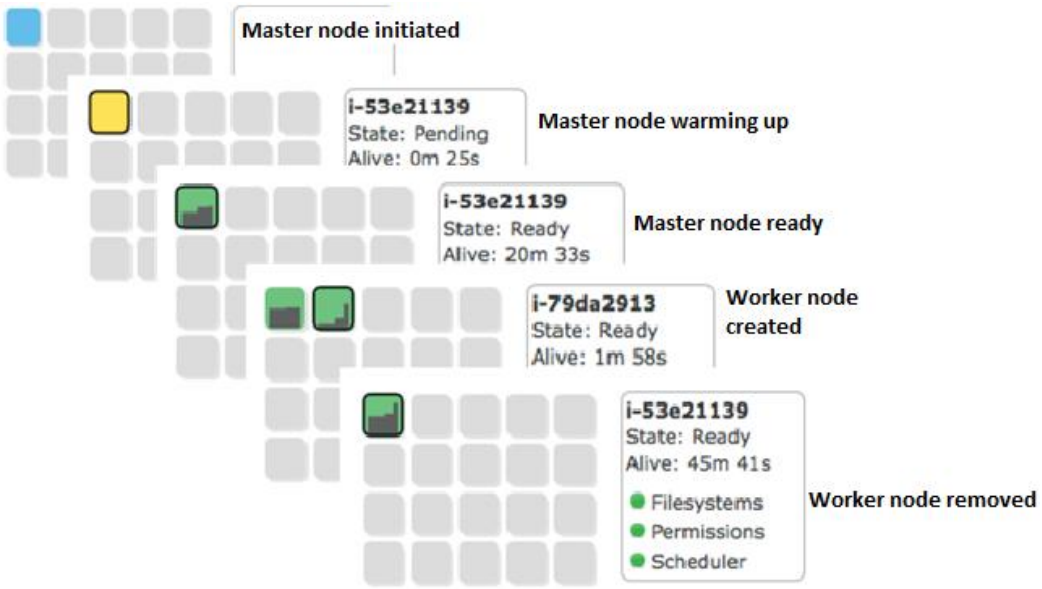

FIGURE-3: Galaxy cloudman interface.

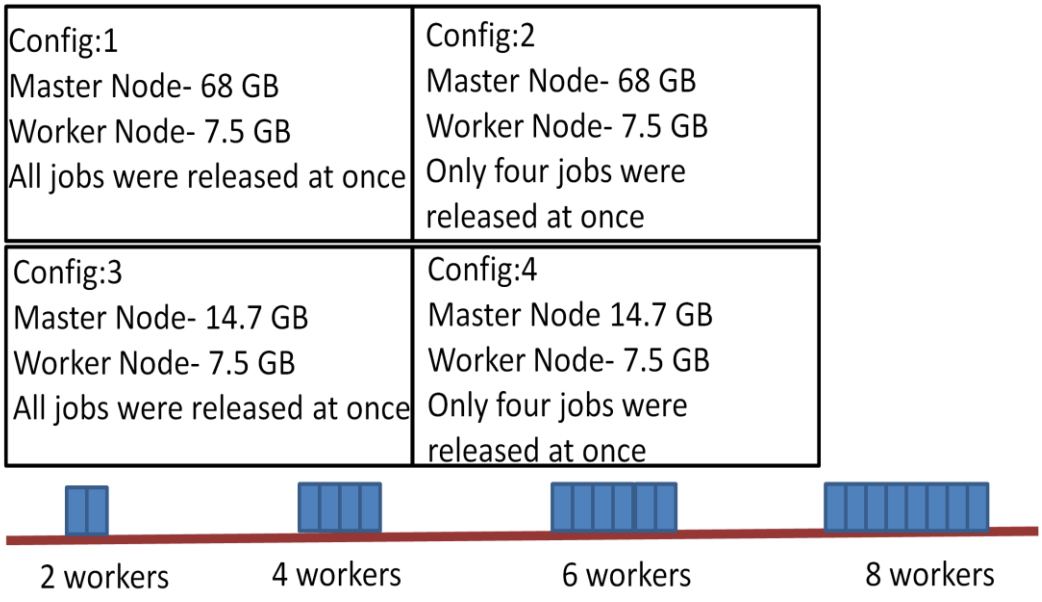

FIGURE-4: Preliminary test Configurations.

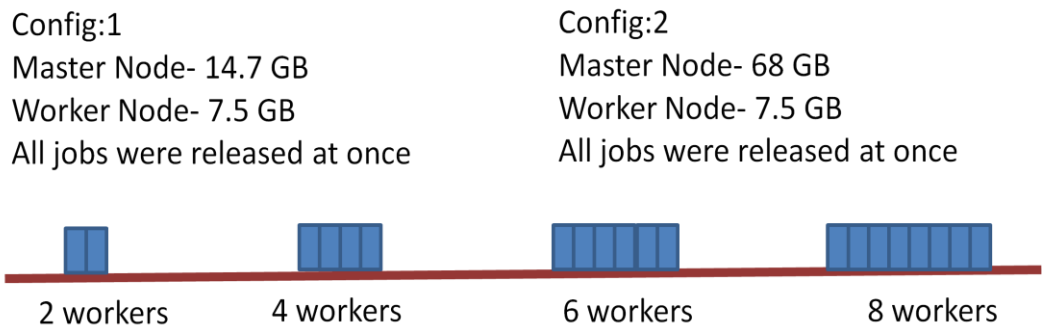

FIGURE-5: Final configurations

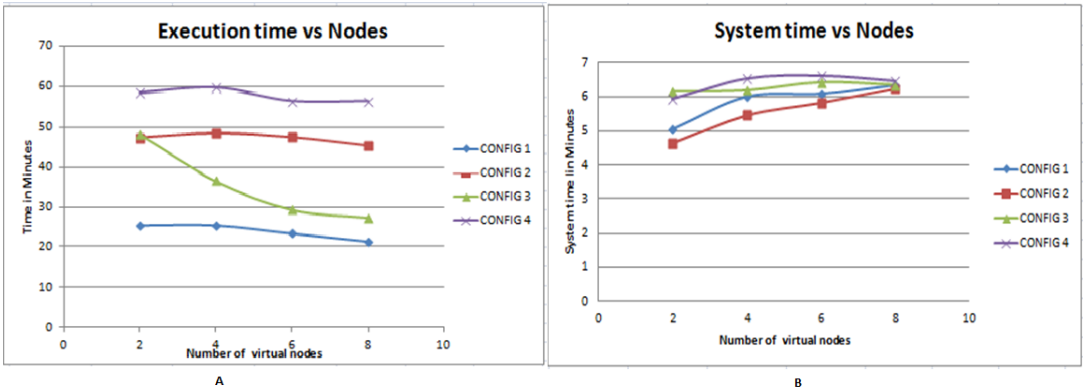

FIGURE-6: Preliminary Tests

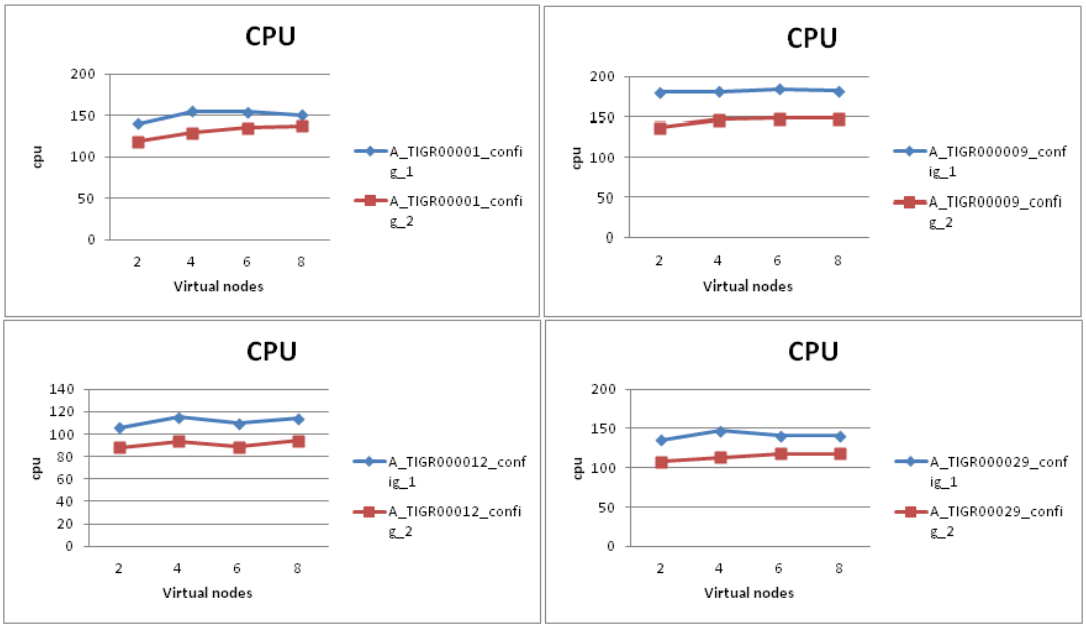

FIGURE -7: CPU versus virtual nodes

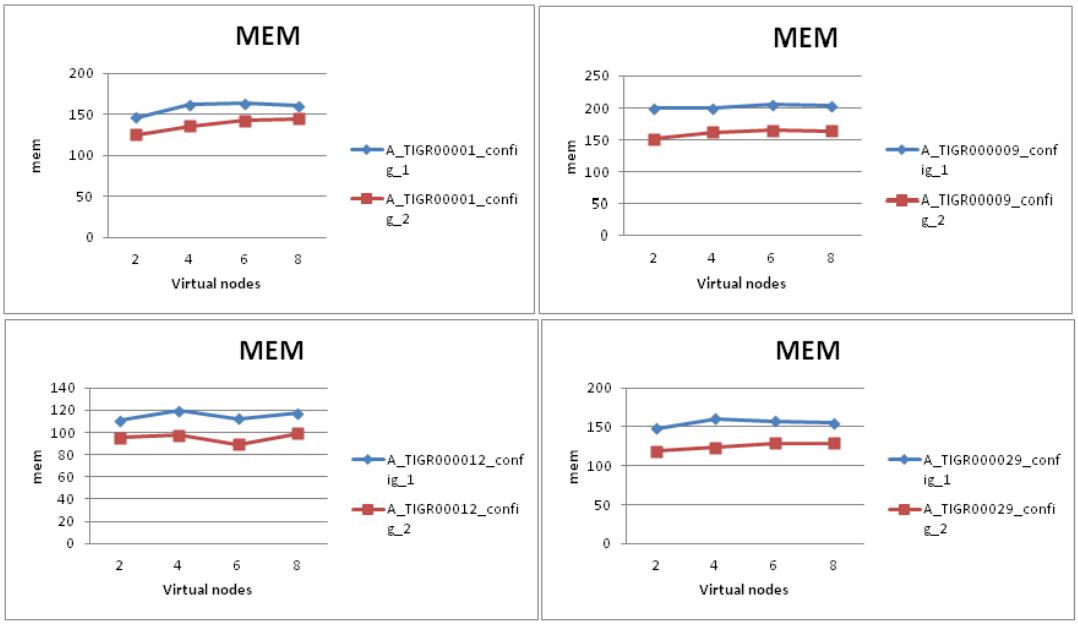

FIGURE-8: MEM versus virtual nodes

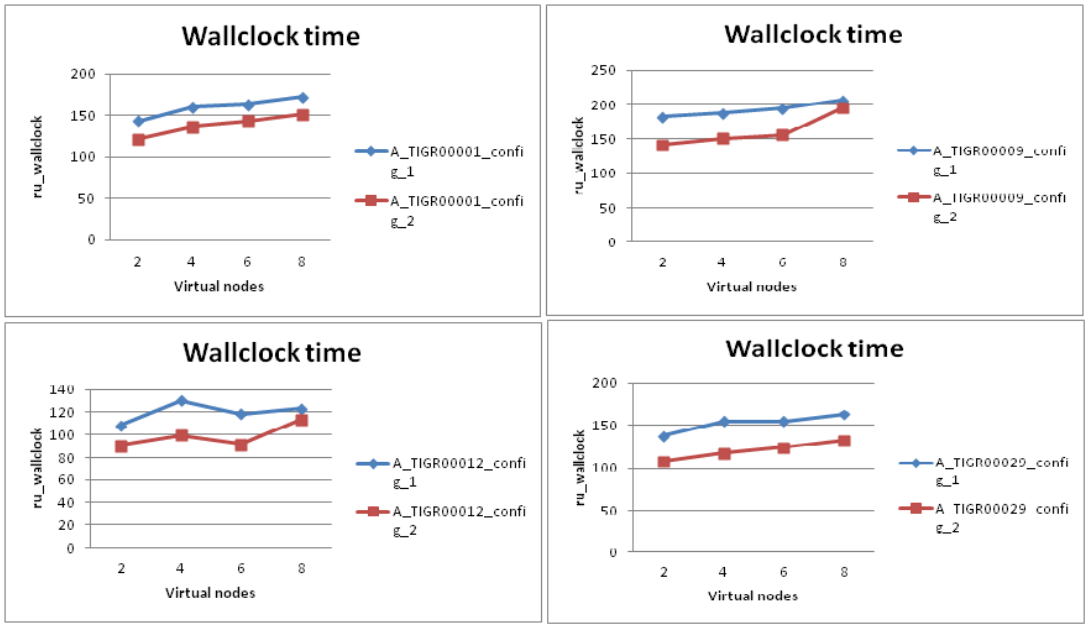

FIGURE-9: Wallclock time versus nodes

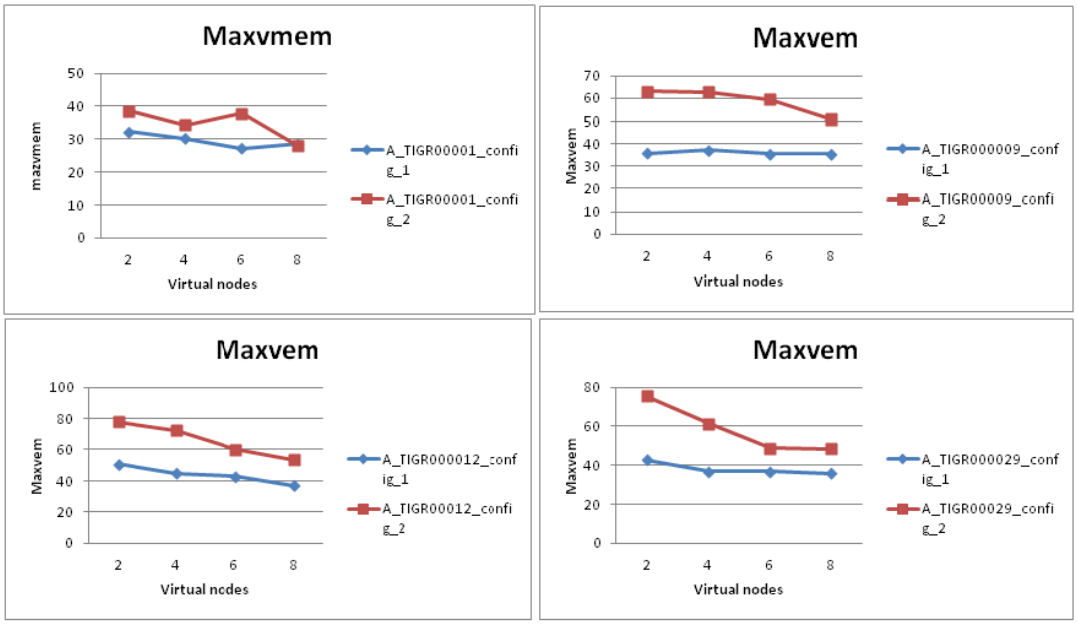

FIGURE-10: Maximum Virtual memory versus virtual nodes

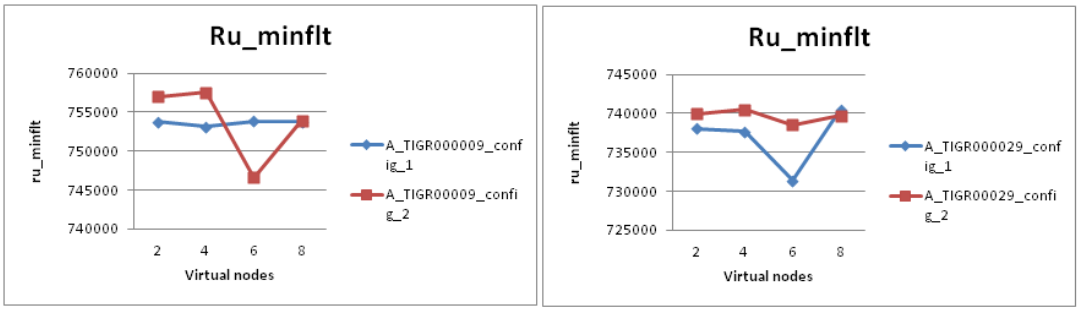

FIGURE-11: Minor page fault versus virtual nodes with HMM

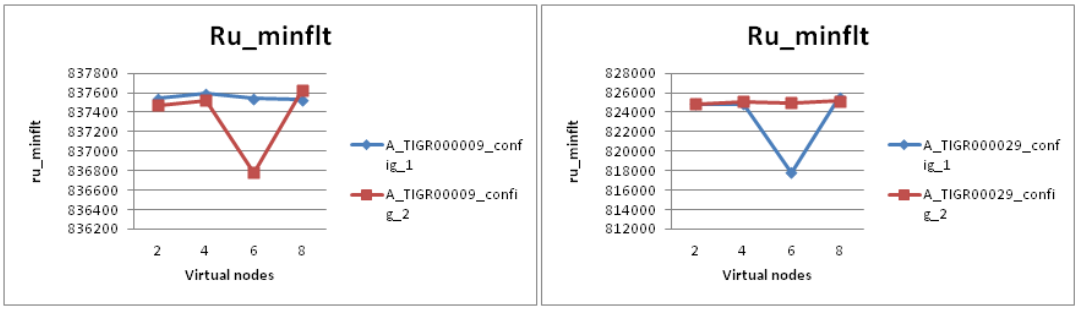

FIGURE- 12: Minor page fault versus virtual nodes with only BLAST