Cloud Computing and Parallel Strategy for Bioinformatics: A Review

Xuan Guo
Department of Computer Science
Georgia State University
Atlanta, Georgia 30319, U.S.A

April 29, 2012

Abstract

This paper provides an overview of the application of cloud computing in certain bioinformatics tasks. Current bioinformatics applications demand both management of huge amounts of data and intensive computation. Cloud computing provides a promising new approaches for bioinformatics tasks that leverage the hardware and software investments on large scale data centres. In this paper, first of all, different bioinformatics tasks are described along with their basic features, such as gene sequence analysis, gene mapping, deoxyribonucleic acid (DNA) fragment assembly, gene finding, microarray analysis, gene regulatory network analysis. The relevance of using parallel strategy and cloud programming models to these problems is then mentioned. These are followed by different applications in cloud computing, along with their merits, for addressing some of the aforesaid tasks. Comparable performance and efficiencies for the applications have been exhibited in cloud service based utility and the managed parallelism. Finally, some limitations of the current research activity are provided. An extensive bibliography is included.

Contents

1 INTRODUCTION 2

2 BASIC CONCEPTS IN BIOINFORMATICs AND CLOUD COMPUTING TECHNOLOGIES 3
  2.1 Basic Units of Cell Biology and Bioinformatics Tasks 4
  2.2 Cloud Computing 5
    2.2.1 Classic Cloud Programming Model 6
1 INTRODUCTION

Over the past few decades, major advances in the field of molecular biology have been a result of an explosive growth in the biological information generated by the scientific community. This deluge of genomic information has, in turn, led to a real requirement for sophisticated databases to store, organize, and index the data, and for specialized algorithms and corresponding tools to view and analyze the data. Bioinformatics can be defined as the use of computational approaches to make biological discoveries [1]. Bioinformatics is an interdisciplinary field involving biology, computer science, statistics, and mathematics to analyze genome content and arrangement, biological sequence data, and to predict the structure and function of macromolecules and other compounds. The ultimate goal of bioinformatics is to create a global perspective as well as to enable the discovery of new biological insights from which consistent principles in biology can be derived [2].

Recently, high-performance computing (HPC) have been gaining the attention of researchers for solving bioinformatics problems. Bioinformatics computing involves the construction of mathematical models and numerical solution techniques, which often require a huge number of computing resources to perform large scale experiments. By the
advantages of Cloud based technologies scientists can have completely customize their execution environment through easy access to large distributed infrastructures and thus set up the perfect experiments conditions. Moreover, since the infrastructure is rent on a pay per use basis, scientists can have immediate access to required resources without planning and release them when the experiments have finished.

Even though Grid technologies have been widespreadly used in scientific computing, some issues still make the access to this technology not as easy as depicted, such as that research groups have to submit a proposal describing the type of research they want to carry out, which may lead to a competitive use of scientific Grids, and minor research projects could not get access to them. Other issues are more important in technical perspective: sometimes specific tools and APIs have to be limited on the hosting operating systems or on the services offered by the runtime environment, due to the grid feature, a pre-packaged environment, in which applications will be executed. Sometimes grid could not be elastic enough to fulfil what researcher needs.

By the force of addressing aforementioned problems, cloud computing [3] has been the current emerging trend in delivering IT services. With the approaches of virtualization technologies, cloud computing provides a variety of services from the hardware to the application level, where the service is charging on a pay per use basis. Another important feature is the ability to scale up and down the computing infrastructure under the different requirement of the application and the budget of users, where scientists can benefit. These features make the spectrum of options available to cover any specific need for their research by the needs of scientists.

The rest of the paper is organized as follows: first, we describe the basic concepts of bioinformatics along with their biological basis. An overview of Cloud computing by defining the reference model and the key elements of this paradigm is also mentioned in Section 2. In Section 3, various bioinformatics tasks and different cloud computing algorithms based methods available to address the bioinformatics tasks are explained. Final thoughts and key observations about the future directions of Cloud computing for bioinformatics are presented in Section 4.

2 BASIC CONCEPTS IN BIOINFORMATICS AND CLOUD COMPUTING TECHNOLOGIES

First, we introduce the basic biological concepts required to understand the various problems in bioinformatics, and then we describe briefly the cloud computing platform and capabilities it offers to a computational scientist.
2.1 Basic Units of Cell Biology and Bioinformatics Tasks

Deoxyribonucleic acid (DNA) and proteins are built as long linear chains of chemical components as biological macromolecules. A DNA genome consists of a large sequence of nucleotides, called bases. For example, human DNA sequences have more than three billion bases. In different biochemical processing of living organisms, DNA plays a fundamental role in two respects. First, it is the templates for the synthesis of proteins, which are essential biological molecules for any organism [4]. Secondly, DNA is a medium to transmit the building plans for proteins from generation to generation. Proteins are in charge of structural behaviour.

The units of DNA are called nucleotides, which consist of one nitrogen base, one sugar molecule (deoxyribose), and one phosphate. Four different nitrogen bases are represented by one of the letters A (adenine), C (cytosine), G (guanine), and T (thymine). Each a linear chain of DNA has a complementary strand pairing to the chain. The complementary rule is the ability of the nucleotides to establish specific pairs (A-T and G-C). In 1953, Watson and Crick proposed the construe the double helix of the pair of two linear strands. All the information is carried by each strand and can be copied over and over again by the biochemical machinery.

The units of DNA are called nucleotides, which consist of one nitrogen base, one sugar molecule (deoxyribose), and one phosphate. Four different nitrogen bases are represented by one of the letters A (adenine), C (cytosine), G (guanine), and T (thymine). Each a linear chain of DNA has a complementary strand pairing to the chain. The complementary rule is the ability of the nucleotides to establish specific pairs (A-T and G-C). In 1953, Watson and Crick proposed the construe the double helix of the pair of two linear strands. All the information is carried by each strand and can be copied over and over again by the biochemical machinery.

Figure 1: Various parts of DNA [4]

A gene is primarily consisted of a sequence of triplets of the nucleotides (exons). Introns (non-coding sequence) may also be included within the gene. A coding zone is a template for a protein. Considering for the human genome, only 3%-5% of the genome are coding; i.e., they direct the synthesis of proteins and other biological compounds. A region before each gene in the DNA is called promoter, which indicates to the cellular mechanism that a gene is ahead. For example, the codon AUG codes for methionine and signals the start of a gene. Promoters are necessary for the initiation of transcription which serve as key regulatory sequences. Transcription is a procedure that a gene is converted to ribonucleic acid (RNA). While translation is a process that the RNA are converted to amino acids. In [5], a comprehensive survey of the research was done in this field. In brief, there are three types of non-coding sequences of the DNA (see Fig. 1) as follows:

1. Intergenic regions: Regions are ignored during the process of transcription between genes.

2. Intragenic regions (or Introns): Regions are cut out from the transcribed RNA to generate the building blocks of the genes, referred to as Exons.
3. Pseudogenes: Regions are transcribed into the RNA and stay there, without being translated.

Proteins are a linear chain of amino acids [4], called polypeptides within cells. Amino acid molecules bond are forming peptides with each other by eliminating water molecules. There are 20 different amino acids (or residues), which are represented by 20 different letters of the alphabet. Each of the 20 amino acids is coded by one or more triplets (or codons) of the nucleotides from the DNA. Based on the combination of a series of code, the DNA is translated into a linear sequence of amino acids; i.e., a protein via mRNA (messenger RNA) [4]. For example, the DNA sequence GAACTACACACGTGTAAC is responsible of the amino acid sequence ELHTCN (shown in Fig. 2).

![DNA to amino acid sequence]

Figure 2: Coding of amino acid sequence from DNA sequence 2.

Different biological problems considered within the scope of bioinformatics involve the study of genes, proteins, nucleic acid structure prediction. A broad classification of the various bioinformatics tasks requiring high computation requirement is given as follows.

1. alignment and comparison of DNA, RNA, and protein sequences;
2. mapping on chromosomes;
3. finding and promoter identification from DNA sequences;
4. of gene expression and microarray data;
5. sequence clustering
6. motif discovery

Descriptions of these tasks and their implementation in cloud computing framework are provided in Section 3. Before that, a taxonomy of the cloud technologies based on their programming models is explained.

2.2 Cloud Computing

The general notion of Cloud Computing can be cast to two broad categories, cloud and cloud technologies. Cloud refers to a collection of infrastructure services such as Platform-as-a-Service (PaaS), Infrastructure-as-a-service (IaaS) etc., Cloud is provided by various vendors where virtualization usually is the key technology. Cloud technologies refers
to various cloud runtimes such as Hadoop, Dryad, and other MapReduce frameworks, and also the storage and communication frameworks such as Hadoop Distributed File System (HDFS), Amazon S3, etc. The parallel framework and databases indicate that the developer is explicitly forced to consider the data parallelism of the computation. The system is automatically able to provide scheduling and distribution, when an application is cast into parallel framework. Under the specific framework, the developer have no requirement to know standard concurrency mechanisms such as threads and fine-grain concurrency control. The system runtime deals with many of the hardest distributed computing problems, most notably resource allocation, scheduling, and the transient or permanent failure of a subset of components in the system. The programming model of parallel framework guides the developer towards an appropriate level of granularity. Finally, developers can focus on the application at a suitable level of abstraction since the resources available. In the rest of this section, a taxonomy of the cloud technologies based on different programming models has described.

2.2.1 Classic Cloud Programming Model

Fig. 3 depicts the architecture of the classic cloud Programming model. The classic cloud programming model [6] is a task processing pipeline approach with independent workers. It uses the cloud instances (EC2/Azure Compute) for data processing. For the task scheduling pipeline, it uses a queue of tasks where every message in the queue describes a single task. The developers propose the scheduling queue with tasks, while the worker in cloud instances picks tasks from the scheduling queue. The programming model also provides a simple fault tolerance capability to the system. After the completion of the task, the workers delete the task (message) in the queue only. A task (message) will get processed by some other worker if the task does not get completed within the given time limit. The following part uses Windows Azure to illustrate the classic cloud programming model in detail.

![Figure 3: Classic Cloud Programming Model [6].](image)
Windows Azure is provided by Microsoft [7], which focuses on improving applications in three areas: administration, availability and scalability. An application running in the cloud or in data centre can be divided into logical parts, where windows Azure formalizes these divisions into roles. A role contains a specific set of code, such as a .NET assembly, and the environment in which that code runs. Windows Azure allows developers to create three different types of roles:

- Web role: Web roles are largely intended for interaction with the outside world via HTTP. Code can be created using various technologies, including ASP.NET, PHP, and Java.

- Worker role: Worker role can interact with the outside world in various ways. For example, a Worker role might include codes that are responsible for conversion of videos to a standard format or some other kind of data analysis.

- Virtual Machine (VM) role: A VM role runs an image of a Windows Server operating system as virtual machine, which is uploaded to Windows Azure. Once its stored in the cloud, the image can be loaded on demand into a VM role and executed.

Every Windows Azure application consists of one or more roles. When it executes, an application that conforms to the Windows Azure programming model must run at least two distinct instances of each role. Each instance runs as its own VM, as Fig. 4 shows.

Figure 4: Windows Azure application runs multiple instances of each role [7].

As described earlier, the example application consists of just one Web role and one Worker role. A developer can customize the number of instances of each role. Unlike those high level parallel programming frameworks such as MapReduce or Dryad, Worker Roles are not constrained in how they communicate with other workers. For persistent storage, Windows Azure provides three storage options: Tables, Blobs, and Queues, all accessed via a RESTful HTTP interface.

- A table is similar to database to store a scalable key-value. A table has a capacity to store billions of entities and terabytes of data.
- A Blob [7] is a file-like object that can be retrieved by name of it. The size of blobs can be up to 50GB in the cloud. Each blob supports a massively scalable blob system and is highly durable as the data is replicated in the data centre.

- A Queue [8] is a reliable message delivery mechanism between the compute roles, which can asynchronously communicate via messages placed in the queue. The message in the queue will remain invisible during this timeout period, and will reappear in the queue. This feature ensures that no message will be lost even if the instance which is processing the message failed.

![Figure 5: Illustration of the suggested Azure application model](image)

Figure 5 illustrates the suggested application model, in which one Web Role instance interacts with the web users and communicates work requests to the background Worker Role instances through durable Queues.

### 2.2.2 MapReduce

MapReduce [9] is a popular programming model for processing and performing data intensive tasks on large datasets. MapReduce provides an excellent framework for developing data mining and machine learning applications in data centres, which is first introduced by a Google for handling large scale web content. This is actually an implementation of an old idea from parallel computing and programming languages. It allows programmers to think in a data-centric fashion and focusses on applying transformations to sets of data records. The MapReduce model consists of 2 functions: map and reduce. Any application formulated this way can be paralleled automatically. The MapReduce uses a user defined Map function that takes a pair of key/value and computes a collection of this type of pairs.

\[
\text{map} :: (\text{key}1, \text{value}1) \rightarrow \text{arraylist} (\text{key}2, \text{value}2)
\]

A reduce function maps a key/value-list to a list of values using a similar technique.

\[
\text{reduce} :: (\text{key}2, \text{value}2) \rightarrow \text{list} (\text{value}3)
\]
Each task of MapReduce is carried out in 4 phases: Map, Sort, Merge and Reduce. The Map phase receives a set of key/value pairs. For each pair, the Mapper function generates a result as a pair of key/value. The Sort and Merge phases groups the data to several bins according to different keys in each pair to produce an array, each element of which is a group of values for each key. The Reduce phase works on the sorted array and applies the reduce function on it. The specific functions used in the map and reduce phases are user defined and varies under different application of the model. The overall computation is depicted in Fig. 6.

Figure 6: Computation of MapReduce [9].

The entire architecture (Fig. 7) is grouped to form 2 independent modules.

Figure 7: DataFlow in MapReduce [9].

Phase 1 (Map and Sort) : Each key-value pair is passed to the map function and results are sent to a global buffer. The buffer consists of a definite number of buckets, each one for a different position. If the output exceeds a threshold pre-defined before, the buffer data is flushed on to the secondary storage. Each bucket is sorted in the memory, before the buffered results are written into the disk. Quick Sort is generally used. The data stored on to the disk is thus sorted in the order of key.

Phase 2 (Merge and Reduce) : After the emitted data from Map and Sort, merge operation starts. The results are grouped on the basis of key and the values are clubbed together on a list. In case of collision in buckets, the new value is append towards the end of the value list for that key. Heap Sort is commonly used. The Reduce stage iterates over all keys and applies the user defined reduce function on them. The results obtained are written back to the disk.

MapReduce uses a scheduling mechanism to optimise node utilization, which is both time consuming and error prone as a parallel computing model. It is not possible to
incorporate a MapReduce model for all applications. Communication, Coordination and Synchronisation between the nodes are prime requirements. Most of the parallel programs are asynchronous and it is pretty hard to analyse the interaction between the machines.

**Models of MapReduce**  Three basic execution units under the MapReduce programming model as show in fig 8. The first one is only using map phase. The second one has standard map phase, sort/shuffle phase and reduce phase. The third one is a iteration of the standard mapreduce model.

![Figure 8: Three basic MapReduce Models [9].](image)

**Runtimes of MapReduce**  In table 1, it shows 5 different MapReduce runtimes. Half of them support C/C++, while the half support Java. The infrastructure of Mars is graphics processors. The first 4 runtimes is a standard MapReduce model, although they can be modified to be capable to execute iterative MapReduce framework. The Twister from Indiana University naturally supports the iterative MapReduce.

<table>
<thead>
<tr>
<th>Runtimes</th>
<th>Description</th>
<th>Language Support</th>
<th>Developed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>MapReduce implementation</td>
<td>Java and Other languages are supported via Hadoop Streaming</td>
<td>Apache</td>
</tr>
<tr>
<td>HCE</td>
<td>Hadoop C++ Extension</td>
<td>C/C++, Java</td>
<td>Baidu</td>
</tr>
<tr>
<td>Phoenix</td>
<td>MapReduce implementation aimed for shared-memory systems</td>
<td>C/C++</td>
<td>Standford Univ.</td>
</tr>
<tr>
<td>Phoenix2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phoenix++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mars</td>
<td>Implements the MapReduce framework for graphics processors (GPU)</td>
<td>C/C++</td>
<td>HKUST</td>
</tr>
<tr>
<td>Twister</td>
<td>Iterative MapReduce implementation</td>
<td>Java</td>
<td>Indiana Univ.</td>
</tr>
</tbody>
</table>
2.2.3 Dataflow Programming Model

Dataflow programming model abstracts the process of computation as a directed graph. The vertex represented two entities: the data created during the computation, and the execute function to output the corresponding vertex data. The directed edge between vertices indicates the dependency relationship between vertices. The dataflow programming model comprises of two key roles, the scheduler and the worker. The scheduler is in charge of monitoring the status of each worker, dispatching ready tasks to suitable workers and tracking the progress of each job according to the data dependency graph. It is implemented as a set of three key services:

- **Registry service**: records the location information for available vertex data. In particular, it create a list of indices to record each available vertex data.

- **Dataflow Graph service**: records the data dependency graph for each job, tracking information of the availability of vertices and explores ready tasks. When task is ready, it will notify the scheduler component.

- **Scheduling service**: allocates ready tasks to suitable workers for executing. The master notifies workers of inputs & initiates the associated execution module for each worker to generate the output data.

The worker works is cooperated with each other and scheduler in a peer to peer fashion. Two functions in each worker are to execute upon requests from master and to retrieve the vertex data. Therefore, the worker is implemented as two services:

- **Executor service**: obtains execution requests from the master, receives input from the storage component, sends output to the storage component and notifies master about the availability of the output data for next vertex.

- **Storage service**: is for managing and maintaining data generated by executors and providing it when it is requested. To deal with failures, the storage component can keep data persistently locally or replicate some vertices on remote side to guarantee the reliability and availability.

Workers transfer vertex data in a P2P manner between themselves and send a fetch request to the local storage service when the executor service receives an executing request from the master node. When all the input data is available for the worker node, the executor service builds an instance for the execution module and initialises it with the input vertices and runs the execution. After the workers finishes, the executor service retrieves the result vertex into local storage and notify the registry service. The storage service keeps hot vertex data in memory while writing cold data on the disk. In order to optimize the performance, the worker schedules the executing and network traffic of multiple tasks as a pipeline. Dryad is a distributed execution engine based on dataflow programming model for coarse grain data parallel applications.
**Dryad** combines the MapReduce programming style with data-flow graphs to solve the computation tasks. Dryad is an implementation of extended MapReduce from Microsoft [10]. Dryad considers computation tasks as directed acyclic graph (DAG) as dataflow programming model mentioned above. The data are stored in (or partitioned to) local disks via the Windows shared directories and metadata files and Dryad schedules the execution of vertices depending on the data locality. The academic release of Dryad uses the DryadLINQ API for programmers [11], [12].

Developer needs to design an arbitrary directed acyclic graph to describe the application’s communication patterns for a Dryad application. The communication is expressed as the data transport mechanisms (files, TCP pipes, and shared memory FIFOs). Dryad is notable for allowing graph vertices to receive an arbitrary number of inputs and outputs. While MapReduce restricts all computations to take a single input set and generate a single output set. The overall structure of a Dryad job is determined by its communication flow. It is a logical computation graph that is automatically mapped onto physical resources by the runtime. In particular, there may be many more vertices in the graph than execution cores in the computing cluster. At run time each channel abstract several concrete implementations that use shared memory, TCP pipes, or files temporarily persisted in a file system. As far as the program in each vertex is concerned, channels produce and consume heap objects that inherit from a base type.

A schematic of the Dryad system organization is shown in Fig. 9. A Dryad job is coordinated by a process called the ‘job manager’ (denoted JM in the figure) with network access to the cluster. The job manager consists of the application-specific code to build the job’s communication graph along with library code to schedule the work across the available resources. The job manager is only responsible for control decisions, since all data is sent directly between vertices.

![Figure 9: The Dryad system organization [10].](image)

A name server (NS) is used to enumerate all the available computers in clusters. The name server also provide the position of each computer within the network topology in order to take account of locality for scheduling decisions.
3 BIOINFORMATICS TASKS AND APPLICATION OF CCs

We now describe the different problems and associated tasks involved in bioinformatics, their requirements, and the ways in which computational models can be formulated to solve them. The classified tasks (as mentioned in Section 2.1) are first explained in this section, followed by a description of how cloud computing techniques are applied in solving them. A summary of bioinformatics algorithm is showed in table 2.

### Table 2: Summary of Bioinformatics Algorithm on Cloud Platform.

<table>
<thead>
<tr>
<th>Bioinformatics Tasks</th>
<th>Number of Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment and Comparison of DNA, RNA, and Protein sequences</td>
<td>7</td>
</tr>
<tr>
<td>Sequence Assembly</td>
<td>1</td>
</tr>
<tr>
<td>Gene Mapping on Chromosomes</td>
<td>7</td>
</tr>
<tr>
<td>Interpretation of Gene Expression and Microarray Data</td>
<td>4</td>
</tr>
<tr>
<td>Sequence Clustering</td>
<td>4</td>
</tr>
<tr>
<td>Motif Discovery</td>
<td>1</td>
</tr>
</tbody>
</table>

3.1 Alignment and Comparison of DNA, RNA, and Protein sequences

An alignment is a mutual placement of two or more sequences from which indicates the similarity or discrepancy among sequences. The applications include comparison and prediction of DNA, RNA, protein sequences, and sequence assembly. Generally, alignment methods can be cast into two categories: global alignment and local alignment. Global alignment [13] maximizes the number of matches along the entire length of the sequence between the sequences. Local alignment [14] gives a highest scoring to the most similar part of two sequences. SmithWaterman algorithm is an application of dynamic programming (DP) for efficient and complete comparison of two (or more) biological sequences [14]. A similarity score matrix is also produced to describe how similar the two sequences are.

In bioinformatics, Basic Local Alignment Search Tool (BLAST) [15] is one of the most widely used bioinformatics algorithms in life science applications. The BLAST algorithm can compare primary biological sequence information to discover the similarities between the two bio-sequences. Given one nucleotide or peptide sequence, the BLAST algorithm searches against a database of reference sequences and discovers all the local similarities.
between the query sequence and background sequences. The NCIB (National Center for Biotechnology Information) provides one implementation of BLAST algorithm, termed as blastall. Owing to the huge amount of the pairwise genome alignment operation, it can be very computationally intensive for a BLAST to run an instance. The large size of the reference databases and query output can also lead to data-intensive, because searching computation is usually determined by the size of reference database instead of the length of the query sequence. Considering GeneBank as an example, which is a classical DNA sequence repository, it enlarges its database to double size before 15 months ago in August 2009. It contains 108,431,692 sequences at that time. Besides, memory is also a bottleneck for BLAST, since BLAST will map the whole subject sequence database into the invokers virtual memory space for the sub-sequence searching. The emergence of Cloud computing supplies us another solution to enhance the power of large-scale alignment search to a much larger set of scientists in biology.

Xiaohong Qiu et al. [6] propose a parallel BLAST, named AzureBlast, which is running on the cloud computing platform of Windows Azure. The work-flow of AzureBlast is showed in Figure 10. To parallel run BLAST on multiple compute nodes, they used the query-segmentation data-parallel pattern. Given a number of query sequences as an input file, AzureBlast will first partition the input sequences into multiple files, each file will be sent to one worker instance to do alignment searching. The results will be merged together from multiple workers after each instance complete their processes. Compared with the alternative database segmentation scheme, which parallel the searching by partitioning the database into segmentations and switching multiple threads to search in parallel, the query segmentation is more suitable for the cloud platform as it needs little communication between instances. The former needs the inter-node communication for each query, while the latter presents a pleasingly parallel pattern which is easy to scale on the cloud.

![Figure 10: Architecture of AzureBlast [6].](image)

Massimo Gaggero et al. [16] designed an executable mapper for BLAST by using their own Python wrapper for the NCBI C++ Toolkit and Hadoop Streaming. In this MapReduce version of BLAST, the reducer is trivial. In order to eliminate the need for
an input formatter, one preprocess is to convert sequence datasets to a one-sequence-per-line format. The mapper simply reads one sequence at a time from standard input, calculates its alignment with the given query sequence to reference database and outputs a tab-separated results line.

Andra Matsunaga et al. [17] proposed and evaluated an approach to the parallelization, deployment and management of bioinformatics applications on distributed environment, and give a WAN-based implementation, called CloudBLAST. The parallel strategy for CloudBLAST is straightforward. Given a set of query sequences as input file, Hadoop splits the file into blocks of same size which user can predefine, with certain possibly splitting in the middle of a sequence. Due to this incorrect splitting, StreamPatternRecordReader was developed to interpret correctly the boundaries of sequences, which are passed as pairs of a key/value to the Mapper that run BLAST. The computed results from mapper are stored into a local file, which are retrieved by DFS to combine into a single result as final output. This process is depicted in Fig. 11. The main contribution of the author is to demonstrate the efficiency of bioinformatics application on Internet-connected resources across the University of Chicago and the University of Florida by using the machine virtualization and virtual machines.

![Figure 11: BLASTing with MapReduce [17]. Given a set of input sequences, Hadoop splits the input file into chunks of same size, possibly splitting in the middle of a sequence. StreamPatternRecordReader was developed to interpret correctly the boundaries of sequences, which are passed as keys of a key/value pair to the Mapper that executes BLAST. DFS retrieves all local file outputs combining into a single result.](image)

Dennis P Wall et al. [18] redesigned a typical comparative genomics algorithm, the reciprocal smallest distance algorithm (RSD), under the Amazon’s Elastic Computing
Cloud (EC2) environment. The original algorithm (RSD) [19] roughly contains 4 steps (as shown in fig. 12): (1) employs BLAST to generate a set of hits, $H$, exceeding a predefined significance threshold as a first step, by given a subject genome, $J$, and a protein query sequence, $i$, belonging to genome $I$; (2) then using clustalW to align each protein sequence in $H$ with the original query sequence $I$ to emit two sequences if their alignments total length exceeds a threshold; (3) the codeml program of PAML is used to obtain a maximum likelihood estimate of the number of amino acid substitutions separating the two protein sequences, given an empirical amino acid substitution rate matrix; (4) from the set $H$, to use a reciprocal BLAST against genome $I$ and re-calculate maximum likelihood distance to further determine whether the pair of sequence is true orthologous pair. RSD can be subdivided into two distinct computational steps for the cloud; one for computation of the BLAST, and the other for estimation of evolutionary distance and determination of orthology. Each step requires a separate mapper. (1) The mapper function for the BLAST is to generate a complete set of results for all genomes under consideration. (2) The mapper of the Ortholog computation step utilizes the RSD runner file and RSD to estimate orthologs and evolutionary distances for all genomes under study, by given BLAST results from previous mapper. The storage of BLAST and RSD results used the Amazon S3 storage bucket. The Hadoop Distributed File System (HDFS) was in charge of local storage of genomes, and genome-specific BLAST results for faster I/O when running the RSD step.

![RSD algorithm summary](image)

Figure 12: The reciprocal smallest distance algorithm (RSD) [18].

Rohith K. Menon et al. [20] presented a novel parallel algorithm for constructing the suffix array and the BWT of a sequence leveraging the unique features of the MapReduce parallel programming model. At the beginning, a natural MapReduce programming has been design: (1) a file with every suffix index in the reference string is to every worker machine; (2) the mappers iterate over this file and generate key-value tuples with the prefix of corresponding suffix as the key and index of the suffix as value; (3) the shuffle/sort phase is to gather all of the pairs with the same key; (4) the reducers construct a list of sorted suffixes within the batches and finally output BWT or suffix array. In the natural way, the problem that there is a non-uniform distribution of mers in the genome is solved
by using dynamic ranges to balance the number of suffixes per patch in the sampling partitioner. Another problem that repeat prefixes in the current suffixes are expensive to sort is addressed by a recursive bucket sort that it first sorts the suffixes according to defined length prefix by Quicksort and then scans the list to iteratively sorts blocks of suffixes with common prefix. The analysis of experiment results showed that the time to index the human genome has been reduced from several hours to just a few minutes using as many as 120-cores on the open-source Hadoop implementation of MapReduce.

Luca Pireddu et al. [21] presented a MapReduce workflow to perform read alignment and duplicate read removal, which are typically the first steps in a DNA sequencing workflow. The algorithm can be separated into three steps: pair reads, read alignment and duplicate removal, which is designed with 2 distinct MapReduce iterations. For pair reads, the mapper function generates a key/value tuple containing the fragment id and the read number as key, and sequence and quality as value. Using custom partitioning and grouping functions, in the shuffle/sort phase, tuples only are grouped by fragment id. The reducer function only needs to construct and output the proper record including the fragment id, the sequences and base qualities of both DNA fragments. For the second step, read alignment and duplicate removal, the mapper function performs the read alignment using a new library, libbwa, which is mainly written in C based on BWA, to produces a pair of read alignment records as key for each pair of reads. The reads without suitable alignment positions are eliminated as unmapped reads. The reducer is in charge of duplicate removal according to the rule: both for their first and second reads, if two pairs alignment positions on the reference genome are the same, these two pairs are considered as duplicates. Only reads with the highest average base quality are kept. In the analysis of performance of this new workflow, the CRS4 Sequencing and Genotyping Platform (CSGP) is adopt as deep sequencing of hundreds of individuals. At the CSGP, the new workflow runs with 8 lanes of input (9 × 108 read pairs) have been processed on 128 nodes in less than 4 hours.

Ananth Kalyanaraman et al. [22] presented a MapReduce-based implementation of MSPolygraph called MR-MSPolygraph for parallelizing peptide identification from mass spectrometry data. In systems biology research, identifying the sequence composition of peptides is of fundamental task. One of the most effective ways to identify peptide mass spectra, which are generated by high-throughput proteomic technologies using mass spectroscopy, is to compare the experimental spectra against a database of known protein sequences. Recently, Cannon et al. [33] developed a novel hybrid statistical method within the MSPolygraph framework, combining the use of highly accurate spectral libraries along with on-the-fly generation of model spectra. Since the processing of each spectrum is independent of one another, it is naturally to split the input experimental spectra across map tasks. The input files containing queries, database and spectral library will be partitioned into nearly equal sized chunks. Each mapper function executes a modified implementation of the sequential MSPolygraph code, which aligns the local
queries against the entire database, and optionally also against the spectral library. A list of hits for each of their queries is emitted from map phase. Since the mappers output cover different subsets of queries, the new framework does not require reduce phase. From the analysis of experiments, to match the entire collection of 64 000 spectra, Hadoop implementation finishes this task in 6 h using 400 cores, while the serial implementation of MSPolygraph is supposed to take ¿2000 CPU hours using a state-of-the-art desktop computer.

3.2 Sequence Assembly

The core of genomic remains sequencing. Bunch of applications analyzing sequence include sequencing new species, profiling abundance of each taxonomic unit in an environment sample, arranging the sequences of DNA, RNA or protein to identify regions of similar functions and constructing second or third dimensional structure of RNA or protein based on one-dimensional sequence. The complete genome of sequence of a certain species is still a computational problem, and although the success of Human Genome Project and other sequencing projects came out recently year, in the biosphere, millions of species that we have not fully studied remain a challenge for scientists in biology, computer science, chemistry and physics.

Recently, as the development of second sequencing technologies, commercial DNA sequencing platforms contain the Solexa Genome Analyzer from Illumina, the Genome Sequencer from Roche 454 Life Science, the Heliscope from Helicos, the SoLiD System from Applied Biosystems and the commercialized Polonator. These machines mentioned above are commonly referred as next-generation sequencers and produce shorter sequences reads with length ranged from 35bp to 400 bp. Short reads and high-through massive coverage increases complexity and intensifies computational issues to recover the genuine genome from assembly program.

Assembly software is challenged by two issues: one is ambiguous sequences including repeat area in genuine genome and sequencing error, the other is the ability to perform de novo assembly of millions or even billions of short reads. Cloud computing applications are increasingly being made available for high-throughput DNA sequencing data, there is a need for publicly available algorithms that can enable other translational biomedical research applications, such as large-scale gene set analysis of expression data.

Gunarathne et al. [23] presented two adjusted classical assembly program Cap3 based on different cloud computing models by utilizing Apache Hadoop and Microsoft DryadLINQ. Cap3 [24] is a sequence assembly program, whose idea is to align and merge sequence fragments to determine the whole genome sequences. The Cap3 algorithm calculates the overlaps between the fragments to remove the poor regions of the DNA fragments, then identifies and deletes the false overlaps, casemates the fragments to construct contigs of one or more DNA segments with common sequence and finally through multiple sequence
alignment obtain genuine sequences. To implement a parallel application for CAP3 using DryadLINQ the following approach were used [25]: (i) split input files and balance the size of input data among each node; (ii) a DryadLINQ partitioned-file is created to store data-partitions information in each nodes; (iii) each node calls standalone CAP3 and captures the standard output of the CAP3 program. In the Hadoop implementation of CAP3, parallel CAP3 sequence assembly is embedded in map phase for MapReduce model. Data is shared across the nodes and Hadoop application adopts map tasks to execute the CAP3 program as an individual process on a given input DNA fragments file. Basing on a comparative analysis of performance, two cloud computing frameworks give a similar speed up and time consumption.

3.3 Gene Mapping on Chromosomes

Gene mapping is to determine the relative positions of genes on a reference chromosome, and to measure the distance between them. It can help molecular biologists to further study a genome. A major goal of the Human Genome Project is to describe a series of diagram maps, like functional maps of each human chromosome at increasingly accurate resolutions. Generally, gene maps can be classified into two types: cytogenetic map and linkage map. A cytogenetic map, also known as a physical map, offers a physical map along the chromosome which divides and arranges chromosomes into smaller fragments that can be propagated and characterized correspond to their respective locations on the chromosomes. A genetic linkage map generates the relative locations (order) of specific DNA markers on the chromosome. Particularly, with gene mapping, biologists try to find specific nucleotides, called Single Nucleotide Polymorphisms (SNP), which is related to many genetic diseases. In [26], when specific nucleotides in one of two or more chromosome are observed in at least 10% of the population, the position of this type is defined as a SNP.

Michael C. Schatz [27] designed a new highly sensitive parallel seed-and-extend read-mapping algorithm, called CloudBurst. The goal of CloudBurst is to optimize for mapping single end next generation sequence data to reference genomes. Unlike RMAP which runs in single machine, CloudBurst is a MapReduce-based read-mapping algorithm, which modified RMAP to run on multiple machines in parallel with Hadoop. Like RMAP, it is mapping large size of short reads under a user defined threshold for limited number of mismatches or differences. Due to the MapReduce model, CloudBurst is split into map, sort and reduce phases. In the map phase, each mapper emits k-mers of length s as seeds and keys from both the reads and reference sequences. The read and reference sequences with same k-mers will be gather together as input for reducer. Finally, in reduce phase, each reducer executes end to end alignments along the position where the read and reference sequence share the same k-mers, giver the defined mismatches and indels. The output from reducer can be converted into a standard format text file of the alignments as
the same format as RMAP. Since there is no difference of input and output files between RMAP and CloudBurst, CloudBurst can replace RMAP in a data analysis pipeline with the identical results. Furthermore CloudBurst provides much greater performance taking the advantages of the distributed programming framework MapReduce called Hadoop. The results presented from the original paper shown that CloudBurst has high scalability: the running times increase linearly as the number of reads increases, and with similar linear improvements for sensitive searches over a serial execution of RMAP.

Ben Langmead et al. [28] presented Crossbow, a Hadoop-based software tool, which perform alignment and SNP detection for multiple whole-human datasets per day by combining the short read aligner Bowtie with the SNP caller SOAPsnp. Adhere to the model of MapReduce, the input tuple for mapper consist of reads name which is the key and the reads sequence and quality strings which equal value. In the map phase, the map function is to align short read to reference sequence by Bowtie. A stream of alignment tuples are produced in map phase, where each tuple holds a primary key containing chromosome and partition identifiers, and a secondary key containing the begin location of chromosome. The value part of tuple is the aligned sequence and quality values. In the sort phase, first cluster alignments according to the primary key then sort according to the secondary key. Finally, the reducer executes SOAPsnp program to perform SNP searching. The invocation of SOAPsnp in per reducer is written in a wrapper script with appropriate options from reference partition. In experiments from the paper, from 38-fold coverage of a Han Chinese male genome, Crossbow completed alignments and SNPs searching in 3 hours on a 320-core cluster. The analysis of SNPs generated by Crossbow showed a level of BeadChip consistence to the results achieved by original SOAPsnp program, but Crossbow finished in far less time.

Deok-keun Kim et al. [29] proposed a cloud-computing algorithm by executing in parallel using Hadoop, which can detect SNPs from high-throughput RNA sequence data. The main idea of it is to identify SNPs by carefully considering the alignment method and sequencing errors inherent in real data. The sequential version of this algorithm contains 4 steps: (1) align RNA read data against to reference sequence by using GSNAP [30]; (2) collect all the bases and the quality scores of effective reads from the first step; (3) compute the likelihood of each observed genotype at each position on the reference sequence; (4) classify candidate SNP into two different types according to the number of SNP bases and the ratio of first two highest scores. In the MapReduce version, two preprocesses have been executed before mapper function: (1) upload the reference sequence and read mapping files to HDFS; (2) split the input read mapping files according to HDFS’s block size and file sizes. In the map phase, (3) the map function receives each and every read base of a read sequence with the base quality to emit outputs a stream of tuples containing a primary key, partition identifier, and a secondary key, chromosome number and offset. For the sort/shuffle phase, (4) it bins the tuples according to the primary key and sorts according to the secondary key. Finally, (5) each reduce function receives tuples for a
given partition identifier in sorted order to output a stream of SNP calls. The analysis of the RNA reads and SNP calls are obtained from Sequence Analyzer.

Che-Lun Hung et al. [31] proposed a parallel diversity-based haplotype block partition and SNPs selection method using the Hadoop MapReduce framework. The input file of haplotype block partitioning and selection is haplotype matrix which can be an mn matrix of m observations over n markers (sites). In the map phase, each mapper function computes the diversity scores of each block within the chunk, which is generated from the initial step that the whole matrix is split into n/m chunks. The key, value tuples for each Map are (block start number, block end number), diversity score pairs. In the reduce phase, there is only one reducer, which executes haplotype block selection algorithm, since the selection is a linear time algorithm. The reduce operation scans the longest block by merging blocks with the interesting diversity scores.

Sang-kyun Hong et al. [32] proposed a parallel and robust copy number variations (CNV) detection algorithm using CNV shape [21] software to run on a cloud computing environment. Copy number variation, a form of structural variants in human genome is an event in which a large DNA fragment (≥ 1 kbp) has population differences by duplications or deletions. The CNV detection method on the Map/Reduce environment can be split into two main stages. In the first stage, coverage is calculated from read mapping results, and mean shift and mean slope conversions are performed for this coverage level. In the second stage, the mean shift and mean slope data calculated from coverage data in the first stage are divided into contig units. Mean shift and mean slope distributions are calculated for each contig, and CNVs are detected by calculating the threshold of this distribution. The mapper function in the first stage is to generate alignment by SOAP software. The tuples, emitted by mapper, contain the number of chromosome as key and mapped location and mapped read length as the value. The Reduce function first collects data, then compute the coverage of each partition and using calculated coverage compute and store mean shift and mean slope. In the stage 2, the mapper function divides mean shift and mean slope data into contig units to emit chromosome number and contig ID as key and location, mean shift, and the mean slope conversion value at each contig as value. The reducer function of stage 2 calculate each distribution, using the DetectCNV() function which is generated by our CNV shape algorithm, by given mean shift and mean slope data collected for each contig unit dimension reduction in the analysis of chemical structures.

Yanen Li et al. [33] presented the SeqMapReduce software for parallelizing sequence mapping based on the MapReduce cloud computing model. The SeqMapReduce can be classically cast into three phases. In the initial phase, the target genome is split into segments of the same length to construct tag/seq pair where tag represents the chromosomal location of the segment and the identifier of current pair. In the map phase, algorithm uses seed-and-extension method to map sequence. It first builds a hash table for the sequence reads and search for seed match on the segments of the reference
genome, where each sequence read is stored as a key/value pair in the hash table. Then it uses the pigeonhole principle to scan qualified seed matches. The genome segment will be scanned for extended matches to key/tmp_res pair where tmp_res records the temporary results containing the read and the genome segment, after a qualified seed match is found. In the reduce phase, the reducer extract temporary information and generate the final results as key/final_res pairs where key indicates the distinct read and the final_res indicates all matching genome segments. The running time of SeqMapReduce scale up quasi-linearly to the number of computing nodes available. In the experiments with 32 computing nodes, it took 4.5 minutes to map 6 million sequence reads to the human genome.

Tung Nguyen et al. [34] built a Hadoop MapReduce-based application, CloudAligner, which is able to deal with long sequences for read-mapping. CloudAligner does not have the reduce phase which not like traditional MapReduce model. CloudAligner is designed to accept two types of input files: the read file and the reference file. Before executing alignment, CloudAligner partitions the read file into bunch of chunks of defined size and allocate them to the mappers. The mapper function aligns the reads in the chunks onto the whole reference genome file. CloudAligner has all fundamental feature comparing to a full-featured sequence mapping tool containing mismatch mapping, bisulfite mapping, pair-end mapping, fastq input and SAM output. The analysis of experiment results indicate that omitting the reduce phase can achieve significant improvement in the performance of alignment MapReduce-based tools.

3.4 Interpretation of Gene Expression and Microarray Data

Gene expression denotes the process of expressed genes that the coded information is converted into the structures present containing mRNA, protein, RNA (e.g., transfer and ribosomal RNAs). The transcription and translation are operating in the cell. While not all genes are expressed, under different conditions, expression level of specific gene will correspondingly increase or decrease in the cells. Conventional wisdom is that gene which share or participate in a common cell function tends to have similar expression profiles with each other than if they do not [35]. Microarray technology [36] aims to measure expression levels of thousands of genes the same time. The successful analysis of microarray data could potentially give answer to many unanswered and important questions could potentially be answered. With the biological observation that similar expressed genes are often co-regulated and are participating in the same cellular processes, many clustering algorithm are commonly used in microarray experiments to identify groups of previous mentioned genes. One can thus construct a distance matrix between inter genes from gene expression level matrix. Executing classical clustering algorithm on this matrix one can find that permutation of genes which can minimize the total distance of adjacent genes.
Massimo Gaggero et al. [16] rendered a Hadoop-based implementation to the core GSEA algorithm [37]. GSEA functions have been rewritten in Python and combined with Hadoop Streaming for the MapReduce version. Two preprocessing phases are required to perform in one local machine sequentially: generate N random permutations of the class labels vector and output them into a file with an integer index. To implement GSEA on Hadoop, three different MapReduce tasks have designed. In the first task, the map function parse class label vector permutation from a file pair containing the dataset and the gene set list file, calculates enrichment scores for all the genes in the set and generates a tab-separated stream containing the name and index of gene and the corresponding enrichment score. While for the reducer, the function receives observed and permuted statistics for each gene set to calculate p-values. In the task 2, the mapper regenerates a permutation ID stream as the key. The corresponding reducer will estimate this data stream sorted by ID and compute for each permutation to output a tuple with gene set name, enrichment score, normalized enrichment score, p-value and the direction of permutations and that of gene sets. In the task 3, the mapper will emit a tuple with the gene set name as the key. And the reduce function is to calculate the q-value according to the original formula in core GSEA. From the analysis of results, it showed the parallel implementation is able to run gene sets databases with thousands of elements and thousands of permutations on datasets of more than 10000 genes by 100 patients.

Ben Langmead et al. [38] presented a cloud computing tool, called Myrna, for computing differential gene expression in large RNA-Seq datasets. Under variant computing infrastructures, Myrna can be run in three modes: Hadoop mode using a Hadoop cluster; Cloud mode using Amazon Elastic MapReduce; or Singleton mode using a single computer. All three modes are parallelized on multi machine or processors. The Myrna workflow contains 7 steps: (1) preprocess, store a list of FASTQ files into file system in parallel across input files; (2) align, align reads to a reference genome using Bowtie in parallel across reads; (3) overlap, compute overlaps between alignments and a list of gene interval sets predefined in parallel across alignments; (4) normalize, generate a sorted vector of per-gene overlap counts for each label from alignment information in parallel across labels; (5) statistical analysis, scrutiny counts for each gene and compute and emit a P-value indicating the probability that difference in observed counts are due to chance between groups in parallel across genes; (6) summarize, scrutiny a sorted list of all P-values and construct a list of top genes ranked by false discovery rate sequentially; (7) postprocess, first get rid of all overlap records not belonging to any top genes, then compute per-gene Q-values, a false discovery rate analog of P-values, finally generate a series of output files containing files for information of each top gene, a table including estimated RPKM values for genes in the annotation, P-values, q-values and plots showing the coverage for top genes. The performance on cloud of Myrna shows that it is able to analyze differential expression on 1.1 billion RNA-Seq reads using less than 2 hours for limited cost.
Lu Zhang et al. [39] developed a gene set analysis algorithm, YunBe, for biomarker identification in the cloud, special on the Amazon Elastic MapReduce service. The original gene set analysis algorithm, kipuMarkers, includes two steps: (1) expression data overlay, map expression measurements onto gene sets, like molecular pathway; (2) pathway scoring, calculate a perturbation score for each gene set based on the comparison of distinct activity levels across two sample groups. In the MapReduce version, the mapper function receives two inputs, samples and pathways, which are represented as two matrices, and execute multiplication using two matrixes to emit a tuple with pathway ID as key and activity value as value. The reducer function calculates the perturbation scores and P-values for each pathway as reported in [40]. The running time of YunBe scales up nearly linearly as the number of cores increases over the desktop program performance.

Christian Vecchiola et al. [41] presented an application of the classification of gene expression data by using an Aneka Cloud. The intrinsic parallelism of CoXCS [42] allows for a distributed, and faster, implementation. The original classification the author chose is CoXCS and the architecture of CoXCS is show in fig. 13. Cloud-CoXCS is a Cloud-based implementation of CoXCS that leverage the Aneka Computing cloud to distribute the evolution of the independent populations of classifiers at each of the iterations. The algorithm implemented in Cloud-CoXCS is the same as CoXCS.

![Figure 13: CoXCS architecture [41].](image)

### 3.5 Sequence Clustering

Data Mining is a technique for searching large databases for patterns. It identifies unknown correlations between variables that may be commercially useful. Sequence data mining is one of the important problems in Bioinformatics.

R.Bhavani et al. [43] proposed a parallel hybrid K-means-DE-ACO clustering approach to extract the genomic features to identify the species from its DNA sequence based on DE and ACO to group the feature descriptors using MapReduce. The hybrid MapReduce version clustering method can be separated into three parts: (1) Extraction of Feature Descriptors; (2) Genomic Clustering using DE-K-means approach; (3) Genomic Clustering using DE-ACO-K-means approach. In the first part, the mapper function calculates the count of occurrence of each 3 lettered keyword for every sequence and the reduce function outputs the (1 X 64) feature descriptor vector for each sequence. In the second part, clustering algorithm combining DE and K-means using MapReduce...
model is as follows: 1) the number of clusters Kmax is chosen; 2) the cluster centers and the activation thresholds for each of the cluster centers are randomly initialized; 3) in each generation, the map function writes the closest cluster center and the chromosome to the reduce function while the reduce function applies DE mutation. In the third part, ACO is applied to update the activation threshold to speed up the rate of convergence. MapReduce model for species identification combining DE-ACO and K-means clustering algorithm is similar to the second part. The difference is that the activation threshold is updated by applying ACO. The analysis of experiment results showed that compared to the results of PCA method, which ranges from 67% to 96%, exploring the entire DNA sequence using MapReduce model generates 100% accuracy of the feature descriptor counts for all size of sequences.

Xiao Yang et al. [44] presented a parallel algorithm for hierarchical taxonomic clustering of large metagenomic samples with support for overlapping clusters and gave a cloud based implementation using map-reduce framework. Metagenomics is the study of a population of organisms, like communities of microbial organisms in their native environments, by fragmenting and sequencing their collective DNA. Computationally, we need to address the following two tasks. Given a set of reads R = r1, r2, , rn as input:

1. Identify all pairs (ri; rj) such that F(ri; rj) ≥ t, where F is the chosen similarity function and t is a threshold.

2. Cluster reads based on their similarity degree.

The algorithm can be cast into two phases: edge construction and incremental quasi-clique enumeration. For the edge construction, total five different map-reduce tasks have been elaborately designed. Task 1 and Task 2 identify reads with a common k-mer, and do similarity counting. Task 3 removes redundant edges, and replaces each undirected edge with two corresponding directed edges. In order to bring together information about edges that must be validated and the corresponding reads, mapper in the next two tasks uses the original input data and information about edges delivered by Task 3. Reducer creates a tuple containing read sequence and a list of read ids with which given read will be compared. Then in Task 5 this list can be used to generate another tuple, with pair of read identifiers as a key, and sequence read as a value. To implement the second part of clustering algorithm, three MapReduce tasks have been built. Task 6 performs edge filtering depending on the thresholds. Task 7 takes as input two types of data: the edge tuples that survived filtering in Task 6, and clusters generated in the previous iterations. Because Task 7 may generate identical clusters, or clusters sharing the same vertex set but different edge set, these clusters are merged by keeping the vertex set intact, while taking the union of edges in Task 8. All three tasks are called repeatedly to obtain the clustering at different similarity levels.

Chunyu Wang et al. [45] provided a k-mer based MapReduce algorithm for Expressed Sequence Tags (ESTs) clustering in large dataset on commodity computers, and imple-
ment the algorithm in mrClust package. The main idea of the algorithm is to use a similarity of two sequences, which is a function of the number of k-mers shared by two sequences, to cluster the EST data. The algorithm needs two iterations of MapReduce to complete the k-mers set and merge pairwise matches. In the first iteration, the mapper function receives the input EST sequences and emits key-value pairs in which k-mer itself is the key and offset information is the value element containing the sequence id and offset position. The reducer function is to write the value list in the order of sequence id to HDFS. In the second iteration, the mapper function is to emit all combinations of sequence ID sharing the same k-mer with the pair of sequence ID as the key and corresponding pair of offset position as the value. The reducer function firstly trims the k-mer match list, and then counts matches by employing a window with length w. If the count reaches a predefined threshold, the reducer generates a local length-w window match as a tuple of two sequence ID. After two iterations, a post-process of mrClust outputs the final result by merging all similar sequence based on ID pair tuples.

Jaliya Ekanayake et al. [6] presented a pairwise Alu sequence alignment application based on cloud technologies Apache Hadoop and Microsoft DryadLINQ. The Alu clustering problem [46] is one of sequence clustering. Alus represent the largest repeat families in human genome. There are about 1 million copies of Alu sequences in human genome, in which most insertions can be found in other primates and only a small fraction (7000) is human specific. This indicates that the classification of Alu repeats can be deduced solely from the 1 million human Alu elements. Alu clustering can be viewed as a classical case study for the capacity of computational infrastructures because it is not only of great intrinsic biological interests, but also a problem of a scale that will remain as the upper limit of many other clustering problems in bioinformatics. Two highly parallel traditional MPI applications, i.e. MDS (Multi-Dimensional Scaling) and Pairwise (PW) Clustering algorithms in Alu application as fig.

![Figure 14: Pipeline for analysis of sequence data [6].](image)

The two cloud based Alu clustering focus on the pairwise SW-G part. A DryadLINQ application to perform the calculation of pairwise SW-G distances for a given set of genes by adopting a coarse grain task decomposition approach. This approach performs minimum inter-task communication and hence ameliorates the higher communication and synchronization costs of the parallel run-time. An Apache Hadoop version of the pairwise distance calculation program based on the JAligner program, the java implementation of the NAligner code used in Dryad implementation.
3.6 Motif Discovery

In complex network, network motifs [47] can be seen as the basic building blocks of specific pattern of local interconnections with potential functional properties. To analyse motifs, Pattern finding in a complex network is the first and most important step. In the pattern finding area, many problems are consider as NP-complete, such as determining graph isomorphism and maximum independent set [48]. Therefore, the pattern finding algorithms always require high time-space complexity. Moreover, when the size of the pattern is big (usually bigger than 4), the number of the intermediate becomes very large (above millions of items), which makes pattern finding to consume a huge of time and memory.

Yang Liu et al. designed a pattern finding algorithm based on Google MapReduce to improve motif detection the efficiency. MapReduce-based pattern finding (MRPF) framework aims to implement frequent pattern finding on complex graphs based on Hadoop. It uses the generation of a canonical label described in [49] to check graphs for isomorphism. After receiving a dataset of a network, MRPF uses one MapReduce task to parse the dataset and form three information tables. Another MapReduce task is executed to extend matches that are subgraphs of the network from size i to i+1. After all matches of patterns of size i+1 have been obtained, the frequency of new patterns will be computed.

Step 1: Distributed storage. In Step 1, the target network is represented as textual files in a specific format. The file can be split into a set of blocks with the nearly same size and allocated to computing nodes.

Step 2: Neighbour vertices finding and pattern initialization. In this step a MapReduce task has two functions, one is to find adjacent neighbour of each vertex to build an adjacent vertices table (Adj_Table), the other is to search patterns of size two (one edge and two vertices) and their matches.

Step 3: Pattern extension. This step also takes one MapReduce task. The map phase extends patterns of size i to i+1.

Mapper extend the matches of size i to i+1, compute their patterns and emit a group key with the patterns and matches. Each mapper generates one or more key-value pairs, and the pairs with the same key will automatically be aggregated into the same reducer.

Reducer - remove the duplicated matches by comparing the canonical label of each match and store just one of the same matches. The outputs of reducers are grouped into different files corresponding to the pattern label.

Step 4: Frequency computing. A MapReduce task is used to calculate the number of the support value of all patterns that appear in the big simple graph.
4 CONCLUSION

The increasing availability of annotated genomic sequences lead to large-scale analysis of complete genomes, the introduction of computational genomics and proteomics and the proteins that they encode for relating to diseases. The reasons for applying computational methods to facilitate the understanding of various biological processes mainly from two of the following perspectives:

• to provide a more global view in experimental design;

• to generate testable hypotheses regarding the function or structure of a gene or protein of interest by identifying similar sequences in better characterized organisms.

In this paper, we have discussed the potential opportunities and the current state-of-the-art of cloud computing for bioinformatics problems. The adoption of cloud computing as a technology and a paradigm for the new era of computing has definitely become popular and appealing within the enterprise and service providers. It has also widely spread among end users, which more and more host their personal data to the cloud. For what concerns bioinformatics computing, this trend is still at an early stage. A various distinct bioinformatics tasks implemented with cloud technologies have been reviewed in this paper. The parallel strategy behind these cloud softwares show that some of these algorithms share the common approach to using popular cloud programming models. For certain sophisticated algorithm, a serious conversion and transformation of input data and procedures of original method is required. Detailed critical analysis of current developments in each of the aforementioned areas of bioinformatics will provide a basis for prioritizing further research for cloud computing and possibly other projects. It seems that such work should improve both areas, that is CC and HPC. Ultimately, it will also benefit scientists that will obtain easier and cheaper access to resources, together with improved tools to take advantage of the new processing power.

References


[37] Aravind Subramanian, Pablo Tamayo, Vamsi K. Mootha, Sayan Mukherjee, Benjamin L. Ebert, Michael A. Gillette, Amanda Paulovich, Scott L. Pomeroy, Todd R.


A Appendix: Classification of Bibliography

A.1 Concepts and Problems in Bioinformatics


- R.B. Altman. Challenges for intelligent systems in biology. Intelligent Systems, IEEE, 16(6):14 - 18, nov-dec 2001. This paper give the current challenges to use intelligent systems in biology to conclude that the ultimate goal of the eld is to enable the discovery of new biological insights as well as to create a global perspective from which unifying principles in biology can be derived.

- Introduction to Computational Molecular Biology. Boston, MA: Thomson, 1997. This paper introduce some basic biology concepts like DNA et al.

- Annie S. Wu and Robert K. Lindsay. A survey of intron research in genetics, 1996. This paper give the review of the intron research in genetics.

- Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48(3):443 - 453, 1970. This paper proposed the global alignment for the pairwise sequences comparison.

- T.F. Smith and M.S. Waterman. Identi
cation of common molecular subsequences. Journal of Molecular Biology, 147(1):195 - 197, 1981. This paper proposed the local sequence alignment for the subsequence to find the optimal alignment.

This paper presented an sequence assembly program named Cap3.

This paper concentrated on SNP searching problem and gave out a efficient algorithm to do the snp-tolerant detection.

• M. Gerstein N. M. Luscombe, D. Greenbaum. What is bioinformatics? a proposed de
nition and overview of the
This paper review the bioinformatics tasks can concluded that Conventional wisdom is that gene which share or participate in a common cell function tends to have similar expression.

This paper introduce the micrarray technology which can measure the expression of thousands of genes at the same time.

• Aravind Subramanian, Pablo Tamayo, Vamsi K. Mootha, Sayan Mukherjee, Benjamin L. Ebert, Michael A. Gillette, Amanda Paulovich, Scott L. Pomeroy, Todd R. Golub, Eric S. Lander, and Jill P. Mesirov. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression pro
This paper proposed a new gene expression analysis tool, called GASE.

• Francisco Azuaje, Yvan Devaux, and Daniel R. Wagner. Integrative pathwaycen
This paper designed an algorithm to calculate perturbation scores and p-value for the gene expression value.

This paper gave a formation of motif detection problem and present an algorithm to discover the useful motifs in the PPI network.

• Mark A. Batzer and Prescott L. Deininger. Alu repeats and human genomic diversity. 2002. This paper brought the concept of Alu and the importance of Alu repeats in the human genomic diversity.


A.2 Cloud Technologies


• Shane Mainali Brad Calder, Tony Wang and Jason Wu. Windows azure blob. Technical report, Microsoft, 2009. This paper describe the blob communication structure in windows azure.

• Microsoft. Windows azure queue. Technical report, Microsoft, 2008. This paper is a technical report of the architecture of windows azure and give the function description of queue in windows azure.

• Chao Jin and Rajkumar Buyya. Mapreduce programming model for .net-based cloud computing. In Henk Sips, Dick Epema, and Hai-Xiang Lin, editors, Euro-Par 2009 Parallel Processing, volume 5704 of Lecture Notes in Computer Science, pages 417-428. Springer Berlin / Heidelberg, 2009. This paper introduce the MapReduce model which is a popular programming model for processing and performing data intensive tasks on large datasets.

• Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Ulfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI’08, pages 1-14, Berkeley, CA, USA, 2008. USENIX Association. This paper proposed a new platform on cloud infrastructure, named Dryading and do some comparison to the traditional mapreduce platform.
This paper gave the architecture of Dryading, the basic framework of it and analysis the performance of Dryading for scientific problems.

This paper introduce the cloud platform Dryad which is an implementation of extended MapReduce from Microsoft.

A.3 Alignment Problems

• Massimo Gaggero, Simone Leo, Simone Manca, Federico Santoni, Omar Schiaratura, Gianluigi Zanetti, E. CRS, Sardegna Ricerche, and I. Pula. Parallelizing bioinformatics applications with mapreduce. chinacloud.cn, pages 1-6.
This paper designed an executable mapper for BLAST by using their own Python wrapper for the NCBI C++ Toolkit and Hadoop Streaming.

This paper propose a parallel BLAST, named AzureBlast, which is running on the cloud computing platform of Windows Azure.

This paper proposed and evaluated an approach to the parallelization, deployment and management of bioinformatics applications on distributed environment, and give a WAN-based implementation, called CloudBLAST.

This paper redesigned a typical comparative genomics algorithm, the reciprocal smallest distance algorithm (RSD), under the Amazon’s Elastic Computing Cloud (EC2) environment.

This paper presented a novel parallel algorithm for constructing the suffix array and the BWT of a sequence leverage the unique features of the MapReduce parallel programming model.

• Luca Pireddu, Simone Leo, and Gianluigi Zanetti. Mapreducing a genomic sequencing workflow. In Proceedings of the second international workshop on MapReduce and its applications, MapReduce ’11, pages 67-74, New York, NY, USA, 2011. ACM.

This paper presented a MapReduce workflow to perform read alignment and duplicate read removal, which are typically the first steps in a DNA sequencing workflow.


This paper presented a MapReduce-based implementation of MSPolygraph called MR-MSPolygraph for parallelizing peptide identification from mass spectrometry data.

A.4 Sequence Assembly Problems


This paper presented two adjusted classical assembly program Cap3 based on different cloud computing models by utilizing Apache Hadoop and Microsoft DryadLINQ.


This paper presented a mapreduce version of Cap3.

A.5 Gene Mapping Problems

This paper designed a new highly sensitive parallel seed-and-extend read mapping algorithm, called CloudBurst.

- Ben Langmead, Michael Schatz, Jimmy Lin, Mihai Pop, and Steven Salzberg. Searching for snps with cloud computing. Genome Biology, 10(11):R134, 2009. This paper presented Crossbow, a Hadoop-based software tool, which perform alignment and SNP detection for multiple whole-human datasets per day by combining the short read aligner Bowtie with the SNP caller SOAPsnp.

- Deok keun Kim, Jee hee Yoon, Jin hwa Kong, Sang kyun Hong, and Un joo Lee. Cloud-scale snp detection from rna-seq data. In Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on, pages 321 - 323, oct. 2011. This paper proposed a cloud-computing algorithm by executing in parallel using Hadoop, which can detect SNPs from high-throughput RNA sequence data.


- Sang kyun Hong, Jee hee Yoon, Dong wan Hong, Un joo Lee, and T. Bleazard. Parallel cnv detection algorithm based on cloud computing. In Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on, pages 286 - 289, oct. 2011. This paper proposed a parallel and robust copy number variations (CNV) detection algorithm using CNV shape software to run on a cloud computing environment.

- Y Li and S Zhong. Seqmapreduce: software and web service for accelerating sequence mapping. Critical Assessment of Massive Data Analysis (CAMDA) 2009, 2009. This paper presented the SeqMapReduce software for parallelizing sequence mapping based on the MapReduce cloud computing model.

- Tung Nguyen, Weisong Shi, and Douglas Ruden. Cloudaligner: A fast and full featured mapreduce based tool for sequence mapping. BMC Research Notes, 4(1):171, 2011. This paper built a Hadoop MapReduce-based application, CloudAligner, which is able to deal with long sequences for read-mapping.
A.6 Interpretation of Gene Expression and Microarray Data

- Massimo Gaggero, Simone Leo, Simone Manca, Federico Santoni, Omar Schiaratura, Gianluigi Zanetti, E. CRS, Sardegna Ricerche, and I. Pula. Parallelizing bioinformatics applications with mapreduce. chinacloud.cn, pages 1-6. This paper rendered a Hadoop-based implementation to the core GSEA algorithm.


- Lu Zhang, Shengchang Gu, Bingqiang Wang, Yuan Liu, and Francisco Azuaje. Gene set analysis in the cloud. Bioinformatics, 2011. This paper developed a gene set analysis algorithm, YunBe, for biomarker identification in the cloud, special on the Amazon Elastic MapReduce service.


A.7 Sequence Clustering

- R. Bhavani, G.S. Sadasivam, and R. Kumaran. A novel parallel hybrid k-means-de-aco clustering approach for genomic clustering using mapreduce. In Information and Communication Technologies (WICT), 2011 World Congress on, pages 132-137, dec. 2011. This paper proposed a parallel hybrid K-means-DE-ACO clustering approach to extract the genomic features to identify the species from its DNA sequence based on DE and ACO to group the feature descriptors using MapReduce.

• Chunyu Wang, Maozu Guo, and Yang Liu. Est clustering in large dataset with mapreduce. In Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conference on, pages 968 -971, sept. 2010. This paper provided a k-mer based MapReduce algorithm for Expressed Sequence Tags (ESTs) clustering in large dataset on commodity computers, and implement the algorithm in mrClust package.

• Xiaohong Qiu, Jaliya E., Scott Beason, Thilina G., Geo Fox, Roger Barga, and Dennis Gannon. Cloud technologies for bioinformatics applications. In Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS ’09, pages 6:1-6:10, New York, NY, USA, 2009. ACM. This paper presented a pairwise Alu sequence alignment application based on cloud technologies Apache Hadoop and Microsoft DryadLINQ.

A.8 Motif Discovery

• M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. Knowledge and Data Engineering, IEEE Transactions on, 16(9):1038 -1051, sept. 2004. This paper designed a pattern finding algorithm based on Google MapReduce to improve motif detection the efficiency.