UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Big data analysis for metagenomics

Edvard Pedersen

INF-3993
Individual Special Curriculum in Computer Science
September, 2011

Abstract

Metagenomics is a rapidly growing field of research. Novel organisms are discovered at a blazing pace, and the commercial uses of these organisms are just beginning to be discovered, helped by the development of new sequencing machines, which produce massive amounts of sequencing data. This has spawned a need for effective processing and storage solutions. In this paper, we provide a short introduction to the field of metagenomics and some of the computer science challenges this field faces due to the large amount of data to be analyzed, as well as some possible tools and techniques to handle the big data challenges. We present a metagenomics pipeline developed at the University of Tromsø, provide a scalability analysis showing potential problems like storage and fault tolerance, and we provide some suggestions to achieve scalability required for production use.

1 Acknowledgements

I extend my thanks to my supervisor Lars Ailo Bongo for his thoughtful comments throughout this project, my co-advisor Professor Nils Willasen for his help and critique of the biological sections, Espen Robertsen and Tim Kahlke for their help with the pipeline, Jon Ivar Kristiansen for maintaining the cluster, and my mother and sister, for spending many afternoons educating me in biology.

Contents

2 Introduction

Genomics and metagenomics are the studies of the hereditary material of microorganisms, which have allowed us to make plants that are immune to pesticides, create artificial human insulin and given us new treatments for cancer, among many other things. An important tool to achieve such biological insights is the analysis of results from sequencing machines, which produce vast amounts of data that needs to be interpreted computationally. This requires the use of state of the art big data analysis to be done in an acceptable time frame.

Big data analysis is the term for techniques used to do computations on peta-scale data, that include requirements with regards to fault tolerance, distribution and reliability are paramount. Systems that attempt to satisfy these requirements exist, but identifying which tools suit our needs is not trivial.

To facilitate the analysis of metagenomic data, different pipelines are used, which consists of many tools to analyze the data, these pipelines are designed to fit specific research project(s), and are as such highly heterogeneous in terms of the tools and approaches used.

We present a metagenomics pipeline developed at the University of Tromsø, provide a scalability analysis showing potential problems like storage and fault tolerance, and we provide some suggestions to achieve scalability required for production use.

This report gives a brief overview of genomics and metagenomics (section 3), some big data analysis systems which can be used to analyze metagenomic data(section 4) and an overview and evaluation of a pipeline in use currently(section 5), as well as conclusions and some suggestions for further development(section 6).

3 Genomics and metagenomics

In this section we present a brief overview of genomics and metagenomics, which is required to appreciate and understand the computer science problems we encounter in these fields.

3.1 What is genomics?

Genomics[1, 2, 3, 4] is the study of the genes of cells, and is a subset of genetics. We focus on the analysis of sequencing data from these genes, since it is among the most important emerging technologies for studying the structure of the genetic material of organisms, and for example, to understand the function of proteins, cells and organisms. It has already given the scientific community information about genetic diseases, which enabled the creation of new drugs for diseases that were previously more severe. The goal is to continue this development, not only to treat more diseases, but also to create new bioactive compounds for use in e.g. biotechnology.

An overview of DNA

The principle for reproduction for most biological entities is that two specimens combine their DNA[1] to create offspring. This offspring then takes some *traits* from each parent, along with some mutations. If selection (artificial or natural) is in play, some traits will be more desirable than others in the long run, in the case of natural selection, the traits for survival in the wild, and in the case of artificial selection, the traits that people find beneficial.

DNA is a double helix containing four different nucleotide *bases*. In the DNA helix, the four *bases* are arranged in fixed pairs (G pairs with C, and A with T). The order of the bases in the DNA chain determine the *traits* of the organism, and the functions they perform.

To create the building blocks of cells, a segment of the DNA unravels, and RNA is created which corresponds to the DNA segment.

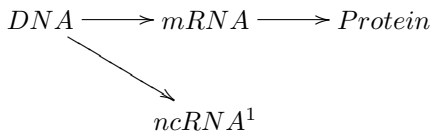

Figure 1: Protein synthesis

This RNA can be one of several different types, depending on the DNA transcribed, where mRNA is the type that codes for different amino acids used in protein synthesis, tRNA is used to connect amino acids to the mRNA, and other types are used for different jobs in the cell.

The importance of the amino acids is that protein is made up of segments of amino acids, and proteins are the primary "workers" inside cells, in that the other components are more or less inert, and the protein works on them. This makes proteins, and enzymes, immensely important parts of life as we know it.

[1] Deoxyribonucleic acid

[1] ncRNA are RNA types other than mRNA, that is, "helper" RNA types, that don't contain hereditary material, including tRNA

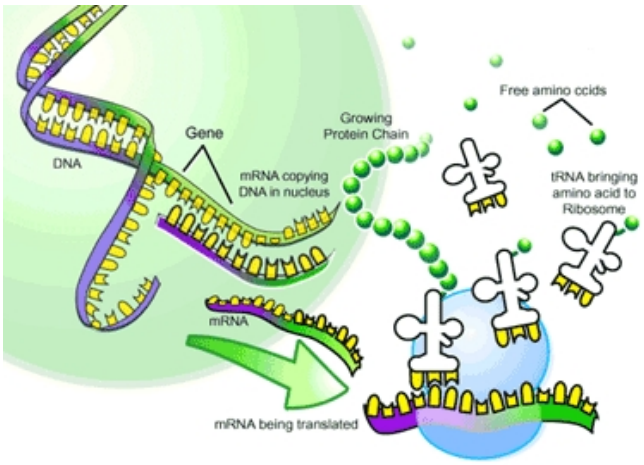

Figure 2: Protein synthesis[5]

So, by understanding the production of these proteins, we can understand what a gene actually does, and use this information to do things like produce human insulin in a lab.

Glossary of terms

Term	Description
Genome	The complete DNA of an organism, which contains both genes and non-protein coding sequences (ncRNA).
Gene	A hereditary part of a genome. Each gene codes for either a protein or an RNA chain that has a function in the organism.
DNA	Living organisms encode their genes in long strands of DNA made up of repeating nucleotide units.
Amino Acid	Building blocks of proteins.
Enzyme	The catalysts for chemical reactions, special proteins.
Protein	Building blocks for cells.
Allele	One of two or more versions of a gene.
Nucleotide	Repeating unit in DNA and RNA; Adenine, Thymine, Cytosine or Guanine for DNA, Thymine is replaced by Uracil for RNA.

DNA sequencing

By understanding the sequences that build up a complete genome, it is possible to perform gene therapy, create crops immune to herbicides, construct bacteria that produce proteins and small molecules and many more.

The key to doing this, is DNA and RNA sequencing. There are a number of technologies and instruments for doing this (described in 3.3), a common denominator for them is the need for processing the raw data produced.

3.2 What is metagenomics?

The primary motivation for metagenomics is that many organisms in nature live in symbiosis with other organisms, which make them hard to grow in a petri dish. Metagenomics[6] avoids this problem by looking directly at environmental samples of whole communities of microorganisms instead of growing the samples in the lab, like it is done within genomics.

Next generation sequencing (discussed below) is an enabling technology for metagenomic studies since it allows us to sequence these microorganism communities as a whole.

To put this into perspective, an environmental sample may have as many as 10 000 different species per gram of sediment, in contrast to conventional genomics, where only one genome is in focus. This is what makes the sequencing difficult, we have a myriad of short DNA sequences[2] from hundreds to thousands of heterogeneous genomes. In addition to assembling the short reads into longer DNA strings[3], we have to annotate the genes to cluster sequences to single genomes and perform diversity analysis.

3.3 Instruments used in genomics

In this section, we take a glance at three widely-used instruments used for genomic research.

Microarray

One of the advances genomics has brought to the world, is genetic testing for disease. This is predominantly what microarrays are used for.

A microarray[7, 8] is an array of samples of genes on a glass slide. It measures the *expression* of different genes. Or in other words, how active the different genes are in creating *mRNA*, which in turn influences their effect on traits, one important thing to note is that the array structure allows a microarray experiment to measure the expression of thousands of different genes in parallel.

This allows us to see things like genetic disease by measuring how active different genes are. An example is Huntington's disease, where patients show reduced activity in a small subset of genes in the brain.[9]

The way a microarray experiment works is this:[10]

1. Different gene sequence strands are fixed into an array of support structures.
2. Complimentary DNA (cDNA) is synthesized from mRNA and a fluorescent tag is added so they can be identified.
3. The cDNA is mixed with the DNA on the array.

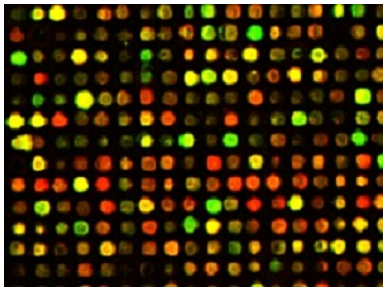

Figure 3: Output of a microarray experiment[11]

[2] DNA reads
[3] Contigs

4. A laser and a camera are used to detect bound cDNA (where the cDNA has bound to the array-fixed DNA)

5. The two resulting images (one for each dye) are combined by a computer (Figure 3), to produce a kind of heat map (a two-dimensional table, with colors representing the expression of each of the two measured strands, typically one diseased strand and one healthy) of the gene expression.

It should be noted that DNA microarrays aren't exclusively used to measure gene expression to find disease and the like, it can also be used for sequencing (by comparing known sequences to unknown ones). Microarrays have, however, been mostly replaced by techniques described in section 3.3, and is not experiencing the popularity growth that those techniques are having.

To summarize, microarrays are both a technology and a methodology for doing genomic experiments and measurements.

Sanger sequencing

Where microarrays are primarily used to measure gene expression, Sanger sequencing is used to determine the sequence of the bases in DNA.

The Sanger sequencing method[12] was one of two sequencing methods developed simultaneously.[13, p. 105] It involves using DNA-polymerase to insert radiolabeled bases to the DNA chain, which stops it from growing further, leaving truncated DNA chains. By running gel electrophoresis, the labeled DNA can be separated. As the smaller sequences will move further than the longer ones, by moving back through the gel, the different bases can be uncovered. With this information, we can interpret the classic illustration of DNA sequencing (Figure 4).

This method of sequencing is in declining use, as it has largely been replaced by next-generation sequencing.

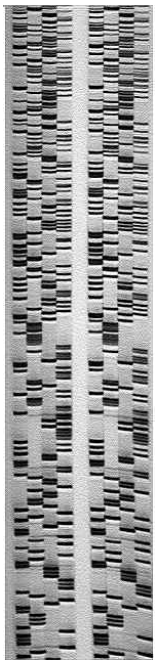

Figure 4: Output of a gel sequencing machine

Next-generation sequencing

The problem with Sanger sequencing is that it is low-throughput, despite generating long read lengths, and therefore incurs a high cost for researchers when performing large-scale sequencing. Many new sequencing methods have been developed, like 454 sequencing[14], Solexa, SOLiD and others.[15] These next-generation sequencing methods allow researchers to sequence large genomes and metagenomes rapidly at low cost.

The main difference between traditional Sanger sequencing and more modern approaches, is that next-generation sequencing methods utilize so-called shotgun sequencing, where many short reads are done in parallel, where Sanger sequencing uses reads of up to 300-1000 base pairs, modern sequencing machines can produce reads of less than 20 base pairs in length[15], but perform them in parallel. In order to assemble these reads into a larger sequences, the reads have to overlap. Several algorithms

have been developed to puzzle them together into contigs, and
the reduced cost of computation has enabled this development.

That being said, there are machines being developed that produce longer reads, which is less computationally intensive to assemble[16].

These advances have lead to a reduced cost of sequencing (which continues to halve every 5 months[16]), allowing researchers to generate more sequences in less time than before.

In summary, the principle behind next-generation sequencing is sequencing many samples in parallel, leading to much faster sequencing, with much larger processing requirements.

3.4 Data wall in genomics

The so-called Data Wall[17, 18] in genomics is the phenomenon that the amount of data produced is exceeding the possibility of storing the data. This means that the results of experiments can not be stored cost-effectively, and poses an issue for researchers where the data they are interested in is simply too large-scale to be stored and maintained reliably.

This problem becomes apparent when we consider that a single sequencing machine can produce 40 giga base pairs (4×10^9bases, equivalent to around 40 gigabytes of pure sequence data) per day as of February 2011[19], and this is doubling every 9 months. When we take into account that the raw output of these machines are gigantic image files, the data produced reaches impractical size, on the scale of 5 terabytes of data per day for some labs[19].

It should be noted that the size of the data decreases as it progresses through the processing stages. The image files are interpreted and stored as sequencing data, for instance, along with additional metadata like quality scores. What data to store, and how much of it, is not an easy question to answer.

In addition to the data from the sequencing machines, the databases of already-sequenced genetic data are huge.

This massive amount of data brings several challenges with it described in section 4.1.

4 Big data analysis

In this section we present an overview of big data analysis, which refers to doing computation on massive datasets, like the ones we find in metagenomics analysis.

4.1 Requirements

When doing big data analysis for metagenomics, we have to have some requirements for what the system should provide.

Provenance

Provenance[20] is a concept of storing the source of data, in addition to the data itself. In the area of scientific data, this means storing the information needed to reproduce an experiment.

This can be a challenge, because it means that the storage requirements increase as we do more processing, as we want to store all the intermediate steps, from the imaging data produced by the sequencing machine, to the final annotated gene data presented to the end-user of the pipeline (see section 5 for a description of the pipeline).

Storage capacity

One requirement we have when doing big data analysis like this, is reliable storage.

We need to be able to store a large amount of data, as the data from next-generation sequencing machines can be huge, and we want to preserve as much of it as is needed to preserve provenance.

Reliability is a prime concern when storing data from scientific experiments like these, this means that the data should not be lost if we lose one machine, or even a complete data center. This means replication.

When we couple these two requirements and look at the amount of data produces by next-generation sequencing machines, we may be looking at storage requirements of a minimum of around 120 gigabytes of sequence data per day running the sequencing machine[4], before doing any computation.

I/O bandwidth

Another requirement is that we need a high-bandwidth network to transfer our data to and from the compute center, between the compute nodes, and to and from the disk. With a large amount of data and relatively low bandwidth, physical transfer of the data may be more efficient than electronical transfer, and even then we may run into problems with the bandwidth of the disk. We also run into the question of reliability when talking about networking, e.g. we don't want a single router failure to stop all communication.

[4] Assuming 3x replication, 40 gbp per day, discarding the raw image data

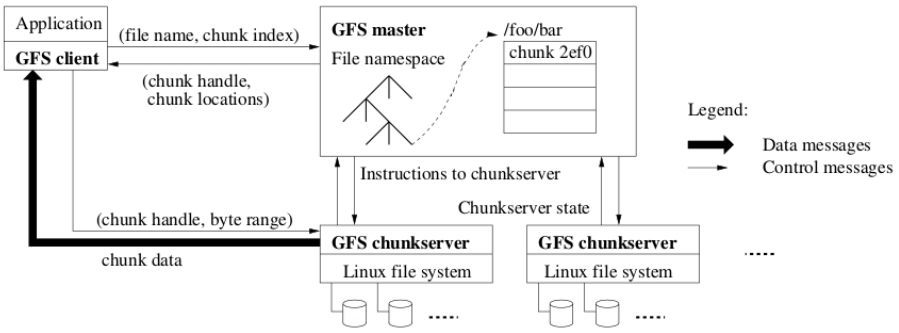

Figure 5: GFS architecture, figure from [21]

Parallelism

A prerequisite for the computations done to be reasonably fast, is that it is possible to run them in parallel. Sequential execution of the computations means that throwing more hardware at the problem won't help. We also want the parallelism to be fault-tolerant, in that we don't want to be stopped by a single failed node, for example. Another requirement here is that we want load balancing, to prevent the issue where one node may be holding up the whole system.

Fault-tolerance

We need our entire system to be fault-tolerant to as large an extent as possible, a malfunction somewhere shouldn't put the system out of action, this is particularly important in big data analysis, as a single task may run for days on end.

Cost

The system needs to be cost-efficient, this means using commodity hardware and not relying on expensive hardware or software to fulfill our requirements.

4.2 Solutions

There are systems used to solve challenges like the ones we have described for metagenomics data analysis, here we take a look at some of them.

Google File System

The Google file system (GFS) is a distributed file system with a focus on reliability and performance[21], designed to run on commodity hardware that will frequently fail. Reliability is ensured through replication, which is achieved by the use of a master server, which itself has it's state replicated, that coordinates the replication amongst the nodes. GFS uses garbage collection to handle deletions and orphaned chunks, this means deleted files aren't actually freed up until some interval has gone by. It is also rack-aware, meaning that data is grouped in a way that minimizes internal communication inside the cluster of nodes. To

support the appending of data, it supports a atomic append operation which allows several data producers to append to the file simultaneously while giving defined behavior.

The main usage GFS is designed for is a collection of large files that are either appended to, or read from, preferably sequentially. This aligns well with a large number of large-scale data processing problems, where we have a lot of data arranged in large files that we want to use for analyzing or processing without changing.

This design is very well aligned with the use of MapReduce-type (described below) systems, which means it is well-aligned with the needs of metagenomic research, since metagenomic work also requires the massively parallel computation capabilities of MapReduce-type systems.

There are several alternatives to GFS-style systems. The most closely related are other network file systems like Amazon's S3, which offer a similar approach to file storage, perhaps the largest difference between S3 and GFS, is that S3 isn't rack-aware, while GFS is. Numerous black-box vendors also exist that sell premade storage systems that are more or less just a black box presenting an interface for file manipulation over the network. The disadvantages to this approach is that it is expensive, and the bandwidth to and from the boxes may limit throughput. Another approach is parallel database systems, which operate on a higher level of abstraction, described in the section on SciDB.

The evaluation in the paper is done using two approaches, a synthetic benchmark and looking at real-world systems. The synthetic testing was done doing ideal read/write operations on a clusters consisting of 16 chunkservers and 16 clients, and shows performance in the realm of 75% of theoretical maximum for reads and 50% for writes. Similar results were encountered in the real-world cluster usage, where the clusters were storing around 50-150 terabytes of data across hundreds of chunkservers. But a caveat here is that the applications running on their cluster are tuned with GFS in mind, so other applications may not achieve the same results.

GFS is in use at Google, while the primary open-source implementation, HDFS (Hadoop File System), is in use in a large number of businesses, examples include LinkedIn, Yahoo! and many others.

MapReduce

MapReduce[23, 24] is a framework for doing parallel computations, the idea is to facilitate SIMD computations done in parallel on a cluster. It was created by Google to help with the massively parallel computations they do in relation to their search engine technology, but has been adopted for widespread use in a variety of fields. It is designed to be a fault-tolerant, simple way to run algorithms on large datasets using clusters.

The usage space of MapReduce is doing massively parallel computations, often in combination with huge datasets, with the caveat that the computations can be expressed in a data-parallel way.

A MapReduce program is divided into two distinct stages, a map stage, and a reduce stage. The mapper functions output into the reducer functions using key/value pairs to decide which reducer the mappers map to. A master server coordinates the logistics of giving out map and reduce tasks to the different machines in the cluster, and reassign straggler tasks to new nodes if needed,

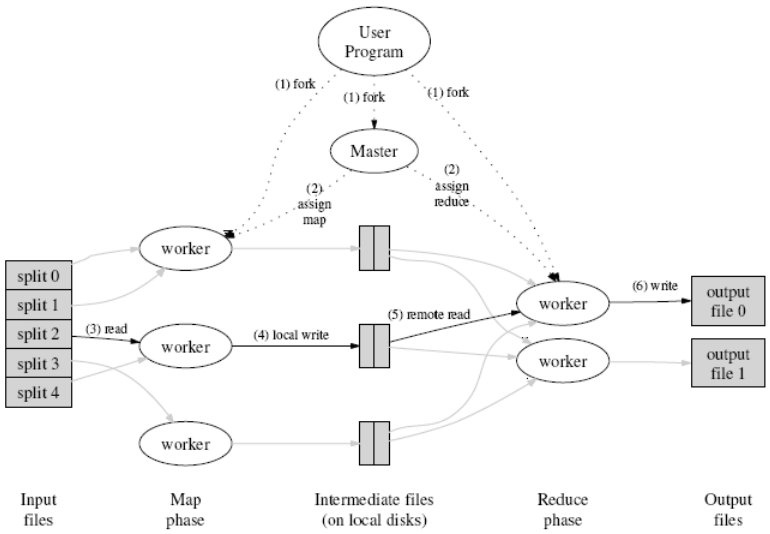

Figure 6: MapReduce architecture[22]

meaning the failure of a node mid-computations does not imply the job needs to be restarted, the master node also handles load balancing and job allocation within the cluster.

There are alternatives that attempt to do similar things, like Dryad[25], which is a more general-purpose framework than MapReduce. There are also more low-level approaches like MPI, and more high-level approaches like distributed databases.

The evaluation done looks at MapReduce programs running on hundreds of compute nodes, running two programs, a grep program which searches through 10^{10} 100-byte records, which runs in 150 seconds, and a sort program which sorts 10^{10} 100-byte records, similar to the TeraSort benchmark. The sort program runs in 891 seconds, which is comparable to the best results for that benchmark at time the evaluation was done. Another indication of the success it has achieved at Google is the number of jobs run using MapReduce, which passed 2 million in September of 2007, using a combined input of 400 petabytes of data.

MapReduce is in use at Google, and the Hadoop equivalent is in widespread use at, for example, Amazon and Facebook.

Pregel

Pregel[26] is a system for distributed large-scale graph processing. The novelty of this system compared to many other graph processing systems is the thought of vertices being nodes of computation, rather than analyzing the graph as a whole. Or in other words, analyzing the graph from the inside, rather than from the outside. Other important properties of the system are fault-tolerance and scalability.

Pregel is used to mine web graphs at Google, which gives an indication of the scale and data it is used to process. It was created at Google, to handle

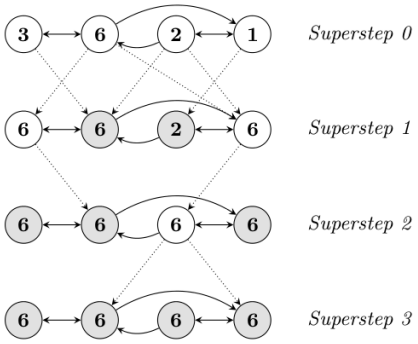

Figure 7: Superstep execution of maximum value, dotted lines are messages, shaded vertices are inactive (voted to halt)

the large-scale graph processing needs they have. It is still primarily in use at Google.

The design of the system is such that a master node coordinates the execution of the job, splitting the graph among the workers. Each worker is responsible for the part of the graph it is assigned, running the code for each vertex. Data is stored using GFS or BigTable (discussed below).

In the system, each vertex is either active or inactive, and all vertices are active at the start. A user-defined compute function is run on each active vertex, and a vertex will go inactive when it has no more work to do unless it is triggered by another vertex in the graph. The compute function is run until all vertices vote to halt (i.e. go inactive), when the job is finished. The execution model is based on the idea of supersteps, so that communication and state changes happen once inside each superstep, and are processed in the next superstep, see figure 7.

There are some alternatives, both Dryad and MapReduce are used in the same problem space for example, even though they don't have the same graph focus, which leads to a more difficult system to use and optimize. There are also parallel graph processing frameworks, but a common denominator is that they aren't fault tolerant in the same manner that Pregel is.

The evaluation in the paper shows that Pregel scales to billions of vertices, where increasing the number of workers from 50 to 800 achieves a speedup of 10. Also tested was increasing the graph size, showing a linear relationship between graph size and runtime. Using random graphs, similar results were obtained.

It is currently in use at Google, Hama, the primary open-source equivalent, is under development for the Hadoop framework.

BigTable

BigTable[27] is a distributed non-ACID columnar storage system, similar to a database, it doesn't implement a relational model though, and instead opts for a lower-level read/write system. The design lies somewhere in between a distributed key-value store and a distributed database, and is designed to scale to petabytes of data and thousands of machines.

The usage space of BigTable is what separates it from GFS, where GFS stores files in a hierarchical manner, BigTable stores data in a three-dimensional

sorted map. This is comparable to the usage of key-value stores. This, combined with in-memory caching, means that BigTable supports random access and low-latency reads, in a way that GFS can't, as well as structuring the data and doing compression.

The structure of BigTable is such that rows are the basic unit of transaction, as opposed to fully relational databases, where operations that change multiple rows are allowed. It's also interesting to note that BigTable is a three-dimensional structure, with rows and columns as usual, but also timestamps. So versioning is supported by design in BigTable. A BigTable system consists of a master server, a number of metadata servers, and a large number of tablet servers. It treats all data as uninterpreted strings, it is up to the user to interpret it. Data is stored using GFS, and Chubby[28] is used to provide locking services. Another important point to note is that data is stored in columnar fashion, instead of being physically organized by rows.

It is highly integrated into the Google ecosystem, and interacts very well with GFS and MapReduce.

In terms of alternatives, distributed key-value stores like Amazons Dynamo[29] are similar in many areas, but the interface is more basic in just maintaining a key-value relationship, and BigTable manages locality by grouping rows, while Dynamo doesn't. Distributed databases are also similar, and also provide a relational data model and transactions, but don't generally offer the benefits of the columnar/locality storage model.

The performance evaluation in the paper is done by running tests on N servers, interacting with a 1786-node GFS cluster, and registering results when N is varied through a number of tests. The tests illuminate the problems with this stack when doing small random reads, where the network is saturated by transferring blocks of mostly useless data (a 64-kilobyte block is transferred for every 1000-byte read), but it also shows good speedup on the other performance metrics (linear reads, reads from memory, writes and scans). Of note is that writes are faster than reads, random reads/writes in particular, by up to an order of magnitude, due to writes being written to a commit log and streamed to the GFS cluster. Even so, perhaps the most compelling performance metric is that BigTable is in widespread use at Google and YouTube, among others.

The open-source implementation used in research environments is HBase, a part of the Hadoop framework.

Dremel

Dremel[30] is an interactive query system for read-only data, built on top of systems like BigTable and/or GFS. It is specifically designed for interactions with the rest of the Google framework, and is in use at Google and elsewhere.

Dremel is used for analyzing very large datasets by using a SQL-like query language that allows the user to do queries on the data. What sets it apart from using MapReduce on BigTable to do these queries, is the time it takes to return results, where MapReduce returns results in minutes, Dremel can deliver the results in seconds, making it useful for getting quick results.

As scientific data often isn't relational, the Dremel framework uses a nested data model instead of a relational one. It also orders the data in a columnar format, meaning that instead of ordering the data according to rows, it orders them according to columns. This requires some extra processing, but it also

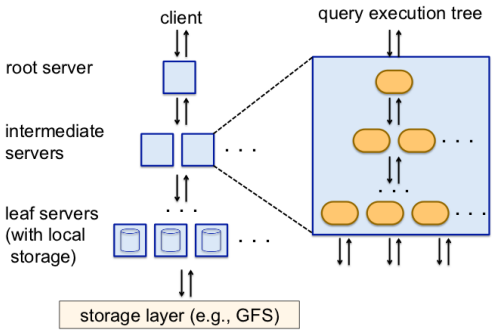

Figure 8: Dremel architecture

means that sparse data sets are possible to process much faster. It should be noted that the read-only nature of Dremel means that it's not useful for data management, only data analysis.

There are some alternatives to Dremel, most immediate is perhaps distributed databases, but columnar storage is not common among distributed databases, and it also uncommon to scale to thousands of nodes. Another alternative is using MapReduce to execute these queries, but a drawback of this approach is losing the benefits of using a higher-level tool to get rapid results, as well as Dremel being optimized to return results in seconds. There is also Hadoop Pig and Hive that can be used to run queries on the Hadoop framework, which is similar in goal as Dremel.

The experiments in the paper show that the large majority of queries in their typical workload execute in less than 10 seconds, and can scan close to 100 billion records per second on their clusters, operating on terabytes of data.

In terms of users, Dremel has been an internal project at Google, but recently they have released BigQuery, which is a service to use Dremel.

SciDB

SciDB[31] is is a distributed data management system in the same vein as traditional distributed database management systems, with a focus on scientific data. The biggest departure from traditional relational DBMS, is the array focus, where data is stored not according to a schema, but in a multidimensional array. This lines up well with data like the output from next-generation sequencing machines.

The architecture of SciDB is such that a central server manages the distribution of data on the nodes in the system. Each node stores the data it contains in columnar fashion, and this is further split into overlapping chunks between servers. It should also be noted that data is immutable in this system.

SciDB can be viewed as a possible alternative to the entire Google/Hadoop framework, as it supports very similar operations, from storage to computations, through an interface familiar to those who have worked with relational DBMS in the past. It is also designed to scale to thousands of nodes. One of the biggest differences between the Google framework and SciDB is the modular approach within the Google framework, versus the package supplied by SciDB. This means that if the requirements align closely with what SciDB offers, it will

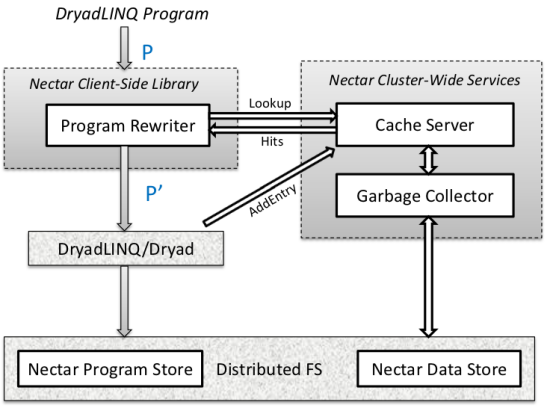

Figure 9: Nectar architecture

probably be a more suited tool to the task than the Google stack, where custom components are required to mix and match pieces to best suit the requirements.

Unfortunately, as SciDB is still in the early stages of development and deployment, there aren't at the time of writing any papers published evaluating SciDB against the Google framework, so it's difficult to say anything about the performance of the system, or the suitability for different problems.

Nectar

Nectar[32] is a system for handling data and computations to reduce storage requirements. It works by storing provenance data and the computations needed to produce the final data for data that isn't accessed often. It is designed by Microsoft Research.

Nectar is useful in a storage-intensive field such as metagenomics, as storing the data from every step of computation is not feasible, and discarding the data without preserving the provenance violates the principles of reproducibility.

This system is realized by associating data with the programs that produced them, or in other words, storing the provenance(see section 4.1) of the data, meaning both the data, and the computation that produced it. This allows the system to discard storage-intensive data that isn't accessed frequently, and recompute it if needed. As a consequence of this, intermediate computations can be re-used, and throw-away results (i.e. results that are only useful for a short time) can be discarded automatically.

Nectar relies on Dryad/DryadLINQ[33] and TidyFS[34] to supply the underlying framework. This is an alternative to the Google/Hadoop framework, where Dryad replaces MapReduce as the computational component, DryadLINQ has a Hadoop equivalent in Pig/Hive, and TidyFS is very similar to GFS/HDFS.

The evaluation in the paper shows the savings for applications analyzing a 1-terabyte document collection. By saving intermediate computations instead of recalculating them, they are able to run 4 different tests using the same preliminary computations, saving over 90% of the computation time per job. There are also some analysis done of 25 production clusters, estimating the computational time saved by using cached results from previous calculations

instead of redoing them, showing that most of the clusters can save 20% to 40% of computation time.

At the time of writing, it is not known if Nectar is in use on production clusters, and the experiments in the paper were done on experimental clusters.

Trident

Trident[35, 36] is a scientific workflow workbench, a tool for managing the type of pipeline described in section 5. This allows researchers to organize the computation on large datasets in a simple and visual manner, while maintaining information about provenance and fault tolerance on a distributed system.

The usage space for Trident is creating and managing scientific pipelines. In a field such a metagenomics, this is a attractive replacement for manually designing and creating the pipelines in script languages.

The system is based on Windows Workflow, which has been modified to suit the needs of scientific projects, like provenance.

Trident abstracts away the underlying storage layer, so that many different storage systems can be used seamlessly. Users can develop algorithms in languages that support .NET and use XML to describe the workflow, or use existing web services, and these can be combined visually in the workbench to create a complete workflow. The workbench also allows users to do things like assign priorities, describe which nodes should be used and scheduling. This data is then used to run the jobs on Windows HPC clusters.

In terms of alternatives, perhaps the most prevalent alternative is using scripts to stitch together the modules that go into a workflow, but this can mean doing a lot of work that has already been done elsewhere. Another alternative is to use the more general workflow managers to design the workflow, but these tools may not have native support for concepts, such as provenance, needed in scientific applications.

At the time of writing, no papers have been published evaluating the performance of Trident, and it is unclear if there are any users.

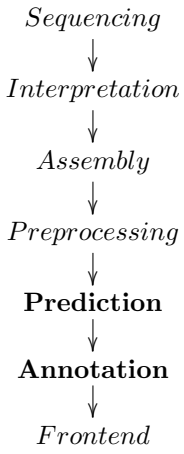

Figure 10: Metagenomic process description, items in bold are handled by the current version of the pipeline

5 Metagenomics pipeline

In this section we present a metagenomic pipeline which is in development by Tim Kahlke at the Willassen lab at the University of Tromsø, and is composed of several modules in a pipelined layout.

5.1 Pipeline description

The metagenomics pipeline is designed to be modular in nature, so that new tools can be plugged in with ease. It is also designed to be a generic pipeline that can be modified to suit different needs.

To achieve this, a generic method for registering tools and databases in a workflow is used. The pipeline then orders these tools and databases in such a way that they execute in the correct order with the correct input.

Adding tools and databases to the pipeline is a relatively simple task, XML files provide the information the workflow needs to use the tools and databases in pipelines. For our pipeline we use a subset of the tools available today, the tools are chosen to be representative of a typical pipeline used in research at the time of writing (as described below).

A typical use case for metagenomics analysis of the system is a researcher, having gathered an environmental sample, sequences the sample. The researcher then decides which tools to use to annotate these sequences, and starts the pipeline using these tools, on the data from the sequencer. When the pipeline is finished, the researcher then explores the data to identify the probable genes in the sample, and what the functions of these genes are predicted to be.

Sequencing, interpretation and assembly

The sequencing step of the process does the actual sequencing, this is the machine that takes the samples, and outputs image data. This image data is then

interpreted to find the millions of short reads, which is then assembled into longer reads (as described in section 3.3). These steps are handled outside of the pipeline by vendor-supplied machines and software.

Preprocessing

The input to the preprocessing stage is the output of the analysis from the sequencing machine.

The preprocessing step is concerned with removing data that isn't interesting, like duplicate sequences, very short sequences or other uninteresting data. This part of the pipeline is currently not implemented.

Prediction

In the prediction step the contigs[5] are analyzed in order to find probable genes. This is done to create input for the annotation step. The prediction is currently done by the Glimmer3 tool.

Annotation

In the annotation step the genes found in the prediction step are annotated with biological information like biochemical function by comparing the genes to existing databases (transferred annotation). To do this, a number of annotation tools are typically used, and the results are combined into a single file.

In our pipeline, we use BLASTp for transferred annotation[6], against different databases, and HMMer3 for functional annotation[7], against the Pfam database.

Frontend

The frontend needs to present the data in a way that is easy to understand and interpret for end users, allowing users to use the data without having to look through the annotations manually. Currently the frontend is not implemented in our pipeline.

5.2 Performance and scalability

5.2.1 Experiment setup

To analyze the performance and scalability of our pipeline, we look at the E. coli bacteria, strain K-12[37] by running parts of it through our pipeline. The complete genome contains 4,639,221 base pairs.

Our main focus is to evaluate the scalability of the system. We are evaluating how data size impacts the performance, how adding CPUs impacts performance, and how using different databases impacts performance.

The cluster we are running our tests on consist of a frontend for running the experiments, and four computers used as compute nodes, each having two Intel Xeon E5620 quad-core CPUs running at 2.4 Ghz. The machines have 24

[5] DNA segments
[6] Comparing the genes to already-annotated genes
[7] In this case used to search for protein domains

gigabytes of memory, three local disks of 1.4 terabytes each, and a network drive of 1.4 terabytes.

Scaling the data size is done by taking the first N bytes (equivalent to about N base pairs) of the E. Coli sequence file. Adding CPUs is done by splitting the input file into N parts and analyzing these parts in parallel. Three databases for BLAST were used in our experiments, all from the UNIPROT-KB collection.

5.2.2 Pipeline implementation

The pipeline (figure 11) consists of 7 steps, executed one after the other, the following step not being started until the preceeding step is completely finished. The pipeline starts out with Glimmer3, which is a gene prediction tool, this is followed by the Glimmer3 exporter, which converts the output from Glimmer3 to FASTA format. Then comes the file partitioner (FileScheduler), which splits up the input file into N parts, where N is the number of threads to be run. Then HMMer3 is run against the Pfam database, for functional annotation, and BLASTp is run against the Uniprot database, for transferred annotation. The annotator follows, which gets the hits from the output of the annotation tools, adds the annotation data to these hits, and combines them into result files. The exporter then combines the annotation data from the annotators into a single result file of a format specified by the user.

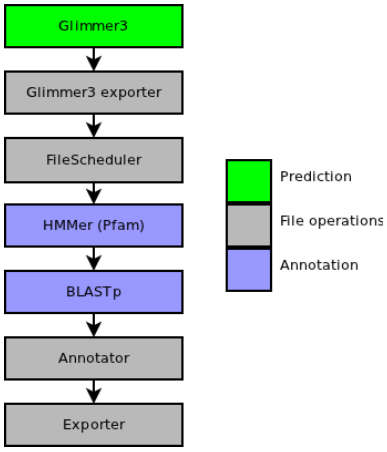

Figure 11: Tools in the pipeline, "file operations" are tools that do I/O work like parsing, file splitting and other non-computational work

Jobs in the pipeline are run using the Sun Grid Engine, which handles the load balancing and scheduling of tasks.

The input to the pipeline is stored on a NFS network drive, and each step of the pipeline copies the file(s) from the network drive to local disk, does the computations, and copies the results back to the network drive. This input consists of genomic data in a FASTA file, which may be output from a sequencing machine, already-sequenced genomes, or subsets of genomes.

The final output of the pipeline is a file containing predicted genes and their annotation, which can be used to locate and identify genes in the input sample.

5.2.3 Results

In our first experiment we measure how the pipeline scales in terms of runtime when the input data size is increased. The experiment was done using a single CPU, and the data size was scaled up exponentially from around 4000 base pairs, to around 128 000 base pairs. Figure 12 shows a roughly linear relationship between input size and runtime, the runtime increased 22-fold when input size was increased 32-fold, from 312 seconds at 4 kbp, to 6805 seconds at 128 kbp. This is expected, as doubling the input data also doubles the numbers

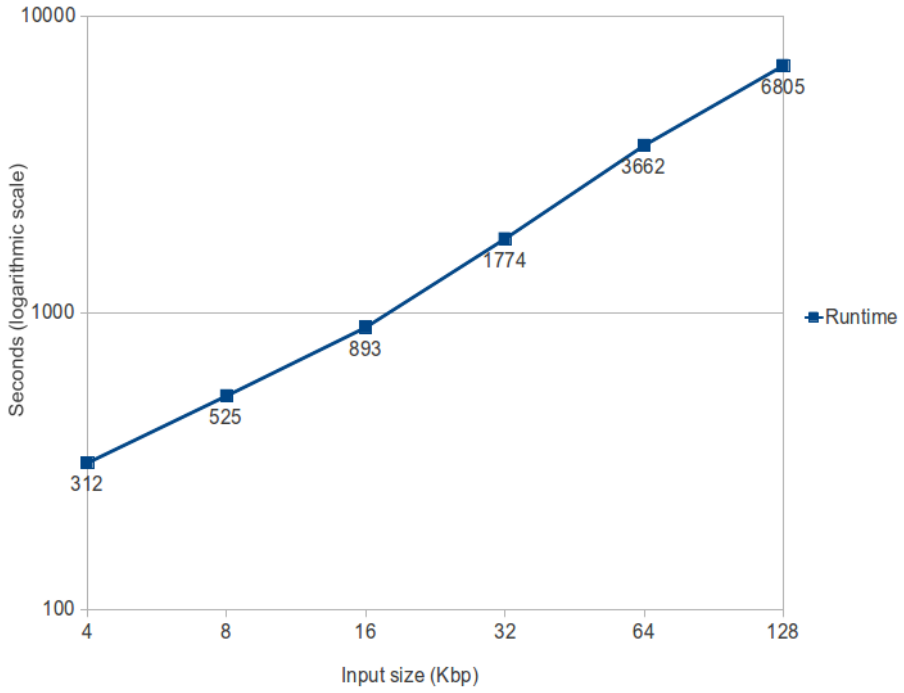

Figure 12: Runtime when scaling input size

of operations that need to be done to analyze the data, we also saw that the runtime is relatively higher at very small input sizes, due to time spent on setup and cleanup in each step. An experiment using 2 560 000 base pairs and 32 CPUs shows that the trend holds true for much larger data sets.

In the second experiment we measure the increase in output size when input size is increased. This experiment was done using the same paramaters as the first experiment, but comparing the output sizes instead of the runtime, this data includes data stored for provenance. Figure 13 shows a linear relationship between input size and output size. The output size grew from 5 mb at 4 kbp, to 151 mb at 128 kbp, and the same test was done using 2 560 000 base pairs and 32 CPUs, showing the same trend by growing to 2.6 gb.

In another experiment we measure how the pipeline handles scaling up the number of CPUs used. The tests were done using the 128 kbp input size from the first tests, increasing the number of CPUs from 1 to 32. Figure 14 shows that the runtime is reduced almost linearly with the number of CPUs used, a speedup of 7 is observed when using 8 CPU cores, while a speedup of 16 is achieved when using 32 CPU cores, we also saw that with 32 CPU cores, some threads finished 48% faster than the slowest threads, spending the rest of the time waiting. This could be due to several threads being scheduled on a single CPU core.

The final experiment evaluates the effect of using different databases for BLAST, and how this effects output size and runtime. The database size (table 1) effects both the runtime and the output size non-linearly. For example, comparing the plants and bacteria databases, we see that increasing the database

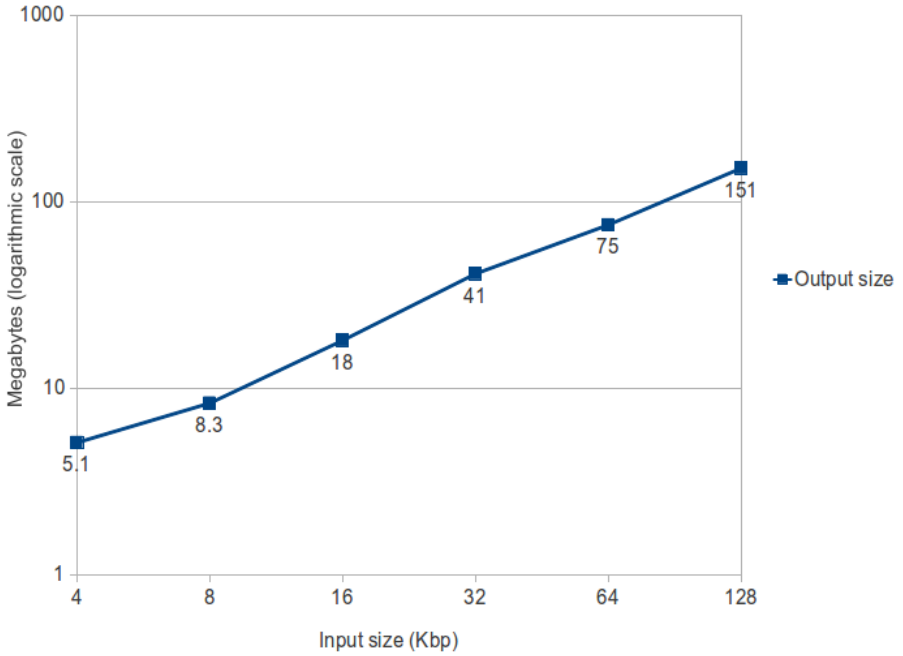

Figure 13: Output size when scaling input size

size tenfold, only increases runtime by around 4 times. This is unexpected, as the amount of data in the database was expected to be linear in relation to the runtime. A possible explanation is that the databases contain genetic information from different domains, and as such contain very different genetic material, but a definitive answer requires more experiments.

Database	DB size (mb)	Runtime (seconds)	Output size (mb)
Bacteria (Baseline)	2852	429	152
Archaea	5	67	25
Plants	312	117	24

Table 1: Effect of on runtime and output size changing database

Our testing shows that three steps are responsible for over 90% of the computation time, these are HMMer3, BLASTp and the annotator. Figure 15 shows how the ratios change in three different tests. We can see that BLASTp takes around 85% of the time for the small inputs, while the annotator takes a larger chunk of time for the large input, meaning that the runtime of the annotator increases faster than the runtime of HMMer3 and BLASTp when input size is increased.

Rudimentary testing of memory usage was also done, and showed that the memory requirements for our tools are not large enough to be a concern at the moment, peaking at around 2.8 gigabytes for BLASTp per node.

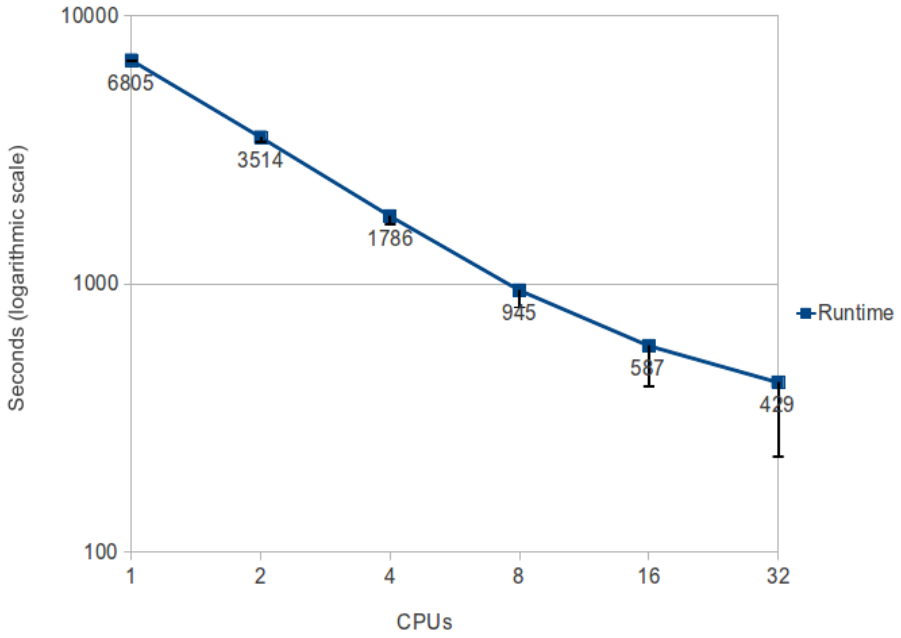

Figure 14: Runtime when scaling number of CPUs, error bars show potential time lost due to load-balancing errors

5.2.4 Analysis

Our experimental evaluation indicates some of the challenges that we face as the input size is scaled up to billions of base pairs.

We have seen that the runtime of most of the pipeline scales well, both when increasing the CPU count and the data size, scaling problems are expected with the annotator when data size reaches giga-scale, there are also problems with the scheduler for many cores, where one core may get more than one task, when there are still idle cores avaliable.

In terms of storage space, we expect the intermediate data to be around 1000 times bigger than the input if using the largest database for BLAST, and the end results to be around twice as big as the input, using smaller databases results in smaller result in smaller storage requirements. The storage space is largely linear with the size of the input in our experiments, provided the database stays the same. The storage space is primarily used for the output of BLAST in our experiments, with the results being around 1/500th of the size of the BLAST output. This means that for our expected data sets (described below), the storage requirements will be around 500 mb for the data set, 5 mb for the input to the pipeline, 10 mb for the results, and 5 gb for the intermediate data, giving us a total of around 5.5 gb.

Data distribution on the cluster is done by copying data from a single NFS disk to a local disk for computations, and back when computations are done. Backups are also done to a network disk.

Varying the database has an effect on both runtime and output size, the relationships appear to be non-linear, but due to the differing data in the databases,

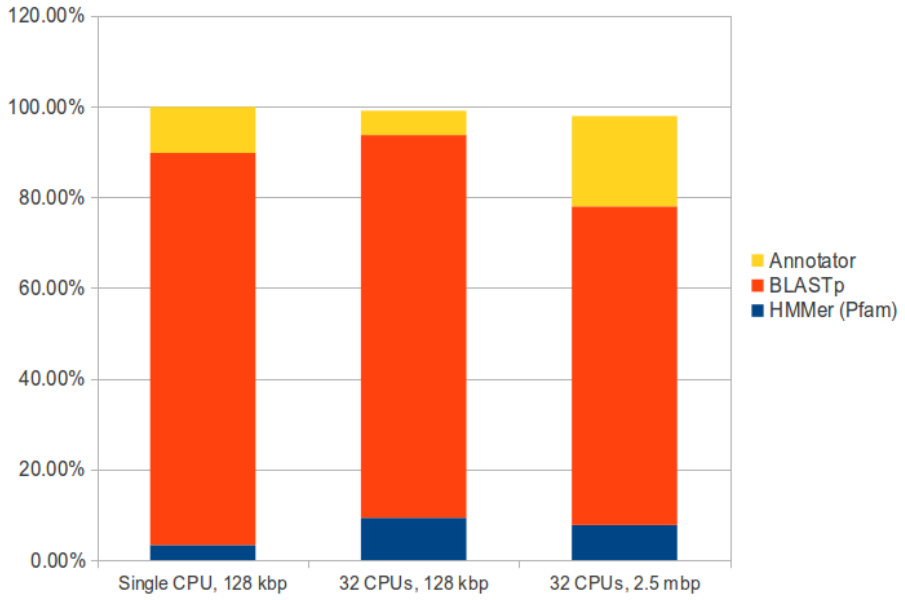

Figure 15: Percent of runtime taken up by each tool

it is difficult to say something conclusive, beyond that the runtime and output size is expected to increase as the databases grow in size. Updates to the database currently require a complete re-run of the experiment.

The fault tolerance of the pipeline has not been tested extensively, but there were occurrences of jobs not concluding successfully due to input errors or being killed, where the pipeline continues to run uninterrupted, while missing input data and therefore not producing correct output.

In total, running experiments on genomes like E. Coli, with a size of around 4 mbp, takes around 4 hours when using 32 CPUs, and produces around 5 gb of data when using the pipeline we have used. Metagenomic work may use much larger input data sets, the latest sequencing machines can produce 600 gb of data in a single run. We expect to be working with around 100 data sets per year. Each data set around 500 mb in size, which will then be reduced by filtering and assembly, leaving around 5 mb to be processed by the pipeline, taking around 5 hours on 32 cores. The pipeline is expected to handle these data sets well, although metagenomic work may require much larger data sets to be analyzed (maybe as much as several gbp), as we process around 10 base pairs per second per CPU core, analyzing a single gbp of data takes around 40 days, so more nodes are required to be able to analyze that amount of data fast.

6 Conclusion and future work

In this report we have provided an introduction to the field of metagenomics, identified some requirements and solutions, and evaluated a pipeline under development at the University of Tromsø. We have found that while the pipeline in question satisfies most of the requirements we have set with regards to the network, parellizability, cost and provenance, it does not fulfill the requirement of fault tolerance.

The storage requirements are largely due to the intermediate results from BLAST, these could potentially be discarded after some time has gone by, but we also want to include more data (tools and databases) in the stored information to preserve provenance. The data is also not stored reliably between backups at the moment, and a system for reliable storage needs to be implemented, GFS may be one part of a possible solution here.

It is expected that the single machine storing the input and intermediate data will become a bottleneck when the data and number of CPUs scale up, high-performance storage distribution may alleviate this problem if it is experienced when scaling up.

Reusing the results from previous experiments may save time when multiple experiments are done using different tools (e.g. using the same input data to different pipelines), an approach similar to Nectar may be useful here.

Branching capabilities should be introduced to the pipeline to enable filtering of unimportant data and more sophisticated pipeline creation, as currently only linear, single-path pipelines are supported.

Changing the pipeline execution so that every part of each step does not need to be completed before the next step begins may yield some benefits, this can be achieved by using a more sophisticated dependency system, where each subtask is only dependent on the preceding subtask, rather than the preceding task, a tool like MapReduce might be used here.

Improving the load balancing may be very useful as experiments show that up to 48% of CPU time of some CPUs is spent waiting for other tasks to complete, closely related to this is fault-tolerance, as the current pipeline is very sensitive to errors, a possible solution to these problems may be using a MapReduce-style tool to handle these individual stragglers.

A related issue is the fault tolerance, which is not complete at the moment. There are ways to improve this, like using more advanced features of the Sun Grid Engine, verifying results, and using a MapReduce style system. This is of particular importance when increasing the number of nodes in the cluster, as it is expected that nodes will fail more often.

There is at the time of writing, no working frontend to the pipeline, but work is being done to use METAREP[38] to display results.

Adding more databases and tools to the framework would allow a broader range of experiments to be run, and this is something that is being worked on.

Analyzing the network and disk bandwidth usage of the pipeline might reveal further improvements, but are outside the scope of this report.

References

[1] Cohen, J. Bioinformatics—an introduction for computer scientists. *ACM Computing Surveys*, 36(2):122–158, June 2004. ISSN 03600300.

[2] Brent, R. Cell : Genomic Biology, 2011.

[3] Wikipedia. Introduction to Genomics, 2011.

[4] Hunter, L. Molecular Biology for Computer Scientists. In *aiiii*, volume 269, pages 1363–4. September 1995.

[5] Wang, J. Transcription and translation. `http://www.scq.ubc.ca/`.

[6] Shi, Y., Tyson, G. W., and DeLong, E. F. Metatranscriptomics reveals unique microbial small RNAs in the ocean's water column. *Nature*, 459(7244):266–9, May 2009. ISSN 1476-4687.

[7] Wooley, J. C., Godzik, A., and Friedberg, I. A primer on metagenomics. *PLoS computational biology*, 6(2):e1000667, January 2010. ISSN 1553-7358.

[8] Allison, D. B., et al. Microarray data analysis: from disarray to consolidation and consensus. *Nature reviews. Genetics*, 7(1):55–65, January 2006. ISSN 1471-0056. doi:10.1038/nrg1749.

[9] Chicurel, M. Gene expression in huntington's disease. `http://www.genomenewsnetwork.org/articles/05_00/huntingtons.shtml`.

[10] NCBI. Microarrays Primer, 2007.

[11] Jessen, W. Treating cancer with personalized medicine. `http://www.highlighthealth.com/healthcare/treating-cancer-with-personalized-medicine/`.

[12] Lander, E. S., et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, February 2001. ISSN 0028-0836.

[13] Watson, J. and Berry, A. *DNA: The secret of life*. Knopf, 2004.

[14] Rothberg, J. M. and Leamon, J. H. The development and impact of 454 sequencing. *Nature biotechnology*, 26(10):1117–24, October 2008. ISSN 1546-1696.

[15] Shendure, J. and Ji, H. Next-generation DNA sequencing. *Nature biotechnology*, 26(10):1135–45, October 2008. ISSN 1546-1696.

[16] Baker, M. Next-generation sequencing: adjusting to data overload. *Nature Methods*, 7(7):495–499, July 2010. ISSN 1548-7091.

[17] Katsnelson, A. and Spencer, N. Human genome: Genomes by the thousand. *Nature*, 467(7319):1026–7, October 2010. ISSN 1476-4687. doi:10.1038/4671026a.

[18] Roos, D. S. COMPUTATIONAL BIOLOGY: Bioinformatics–Trying to Swim in a Sea of Data. *Science*, 291(5507):1260–1261, February 2001. ISSN 00368075.

[19] Kahn, S. D. On the Future of Genomic Data. *Science*, 331(6018):728–729, February 2011. ISSN 0036-8075. doi:10.1126/science.1197891.

[20] Muniswamy-Reddy, K.-K., Macko, P., and Seltzer, M. Provenance for the cloud. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 15–14. USENIX Association, Berkeley, 2010.

[21] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29, December 2003. ISSN 01635980. doi:10.1145/1165389.945450.

[22] University, G. C. Introduction to parallel programming and mapreduce. `http://code.google.com/edu/parallel/mapreduce-tutorial.html`.

[23] Dean, J. and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. ISSN 00010782. doi:10.1145/1327452.1327492.

[24] Dean, J. and Ghemawat, S. MapReduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72, January 2010. ISSN 00010782. doi:10.1145/1629175.1629198.

[25] Isard, M., et al. Dryad: distributed data-parallel programsfrom sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59, 2007. ISSN 01635980. doi:10.1145/1272998.1273005.

[26] Malewicz, G., et al. *Pregel: a system for large-scale graph processing*. SIGMOD '10. ACM Press, New York, New York, USA, 2010.

[27] Chang, F., et al. BigTable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems*, 26(2):1–26, June 2008. ISSN 07342071. doi:10.1145/1365815.1365816.

[28] Burrows, M. The Chubby lock service for loosely-coupled distributed systems. *Time*, 11:335–350, 2006.

[29] DeCandia, G., et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007. ISSN 01635980. doi:10.1145/1294261.1294281.

[30] Melnik, S., et al. Dremel: interactive analysis of web-scale datasets. *Proc. VLDB Endow.*, 3(1-2):330–339, 2010. ISSN 2150-8097.

[31] Brown, P. G. *Overview of sciDB*. SIGMOD '10. ACM Press, New York, New York, USA, 2010. ISBN 9781450300322. doi:10.1145/1807167.1807271.

[32] Gunda, P. K., et al. Nectar: automatic management of data and computation in datacenters. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8. USENIX Association, Berkeley, 2010.

[33] Yu, Y., et al. DryadLINQ : A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages:1–14, 2008.

[34] Fetterly, D. and Isard, M. TidyFS : A Simple and Small Distributed File System. *researchmicrosoftcom*, 2010.

[35] Simmhan, Y., et al. Building the Trident Scientific Workflow Workbench for Data Management in the Cloud. In *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 41–50. Ieee, 2009. ISBN 9781424450824. doi:10.1109/ADVCOMP.2009.14.

[36] Microsoft Research. Project Trident: A Scientific Workflow Workbench, 2011.

[37] Blattner, F. R., et al. The Complete Genome Sequence of Escherichia coli K-12. *Science*, 277(5331):1453–1462, 1997. ISSN 00368075. doi:10.1126/science.277.5331.1453.

[38] Goll, J., et al. METAREP: JCVI metagenomics reports—an open source tool for high-performance comparative metagenomics. *Bioinformatics*, 26(20):2631–2632, 2010.