Towards faster RNA Sequencing analysis

Roman Sirokov

Helsinki October 22, 2014

MSc Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Roman Sirokov

Työn nimi — Arbetets titel — Title

Towards faster RNA Sequencing analysis

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	October 22, 2014	52 pages

Tiivistelmä — Referat — Abstract

Processing data produced by next-generation sequencing technologies is a computationally intensive task. We aim to speed up this task by means of parallel computing. Our paralell computing solution employs Slurm for managing workload between different nodes. It can be used on top of Anduril, a workflow management software for scientific data analysis, as well as on its own. To test the performance of our solution, we use a workflow for post-processing and analyzing RNA-Seq data that originates from lymphoma patients. Data consists of 447 samples independent from each other that can be processed in parallel. To evaluate the performance we employ three different metrics: a level of parallelization, execution time and CPU load. The workflow achieved an excellent level of parallelization for the provided data of 447 samples with the upper bound of 894 cores. Execution times were compared in two different manners: with a set of homogenous samples of various sizes and heterogenous samples. Homogenous samples took on average the similar amount of time regardless of the size of the set. With heterogenous sets the execution time of the largest sample was chosen as a reference and the total execution time was 32% longer than the baseline. Finally, the CPU load of each component was measured. With homogenous sets high CPU load was observed, while with heterogenous sets CPU idling was detected.

ACM Computing Classification System (CCS):
Life and medical sciences, Bioinformatics
Computer systems organization, Architectures, Distributed architectures

Avainsanat — Nyckelord — Keywords
parallel computing, distributed computing, bioinformatics, RNA-Seq, NGS

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Acknowledgements

I take this opportunity to express my profound gratitude and deep regards to Sampsa Hautaniemi for giving my opportunity to work on this thesis in his lab. It was a precious experience, which taught me a great deal and provided a glimpse to the world of science. I also would like to thank everyone at the lab for guiding me, whenever I was lost. In particular I would like to thank Alejandra Cervera for providing insight about her RNA-Seq pipeline. Javier Núñez-Fontarnau for explaining in and outs of Anduril and being overall heplful. Kristian Ovaska for guidance on my thesis and providing valuable feedback. Ville Rantala for having an answer to virtually any question. Thank you guys.

I am obliged to Mikko Koivisto and Juha Karkkaenen for supervising my thesis and finding time to read and grade it.

I am grateful to Veli Mäkinen and Sirkka-Liisa Varvio from the MBI program in the University of Helsinki for teaching me bioinformatics. I want to further thank Sirkka-Liisa Varvio for being my guardian angel throughout studies and putting me back on track, whenever I went astray.

Lastly I would like to thank my lovely girlfriend, Anu Manuli, to whom I say mau-maumau mau maumau mau mau.

Contents

1 Introduction

In the recent years sequenced genomic data has become an enormous aid in virtually any biological research. The sequencing technology has made a significant progress in the last 15 years bringing the run-time and cost of sequencing down. The ready availability of sequenced data has made possible a novel approach to treat diseases and come up with new medicine. For example, in the recent case of the Ebola outbreak, the sequenced data of the virus led to finding the origin and transmission of the virus [1]. A variation of next generation sequencing technology is RNA sequencing (RNA-Seq), which is used for revealing a transcriptome, the set of RNA molecules, as well as its quantities at a moment in time. Among other uses, the transcriptome data provides the ability to look for changes in gene expression levels that sheds light on the activity of particular genes.

Despite all the progress made in sequencing technology, the process is computationally intensive. Data sets measure in terabytes and analysis might take days, if not weeks or months to complete. In order to speed up the analysis, several computers may be employed to process data in parallel. This approach is called parallel or distributed computing. In the heart of a parallel computing solution lies a resource manager, a piece of software that coordinates execution of jobs between different computers. Another essential piece of parallel computing is an ability to share data between computers.

The goal of this work is to speed up processing of RNA-Seq data using distributed computing methods. Our implementation is built on top of Anduril, a workflow system for scientific data analysis, but also can be used on its own. We evaluated a number of resource managers, as well as distributed storage mechanisms to make a distributed computing solution accommodating our needs. As a case study we use an RNA-Seq workflow for Anduril that is designed to process raw RNA-Seq data into a more usable form and convert it to gene expressions matrices. The RNA-Seq data in question comes from patients suffering from lymphoma, a type of blood cancer and the ultimate goal of the case study is to determine which genes are responsible for facilitating the disease.

The thesis is organized as follows. Section 2 provides necessary biological background information, details on Anduril, an overview of distributed computing, as well as an evaluation of several resource managers and shared storage systems. In Section 3 we provide technical details about the implementation with usage examples. Section 5

evaluates the performance of the RNA-Seq workflow run using a number of threads, running time and CPU load as metrics. Finally, in Section 6 we discuss performance of the solution, as well as possible directions for future research.

2 Background

2.1 DNA Sequencing

The building block of life is deoxyribonucleic acid or DNA, a molecule that encodes genetic instructions that define a living organism. DNA molecules take a form of a double-stranded helices that are made of a sugar phosphate backbone that has building blocks called nucleotides or bases attached it (Figure 1). Each nucleotide can take one of four forms, adenine, cytosine, guanine and thymine that are encoded as letters A, C, T, G. The strands are connected together using a process called complementary base pairing, such that the A base always pairs T and the G pair with C. Thus one strand always mirrors the other one and the combination of two nucleotides is called a base pair.

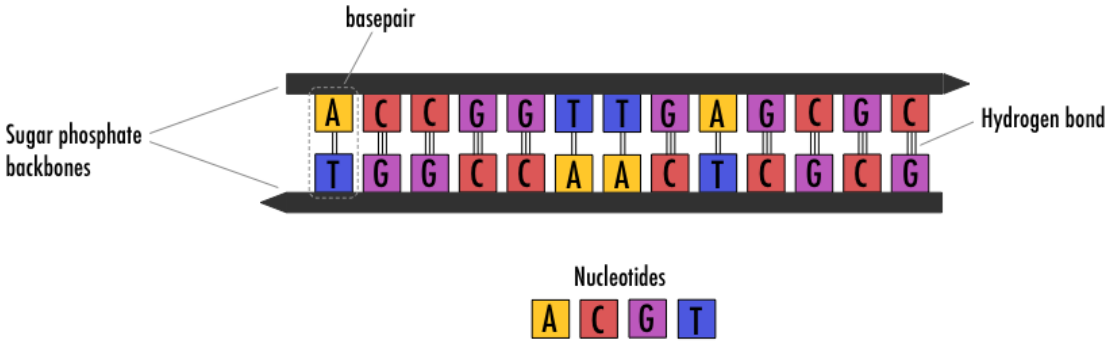

Figure 1: Structure of DNA.

The process of DNA sequencing determines the sequence of nucleotides from an organic tissue of an organism. Sequencing is a method that transforms data stored in DNA molecules into data that can be read and processed by computers. DNA sequencing takes root in the 1970s, when the Sanger sequencing was introduced [2]. The technology is based on a chain termination method, in which a DNA sequence is copied repeatedly producing fragments of different lengths. This is achieved by special nucleotides called chain terminators. When encountered they stop DNA polymerase from further copying. Thus a large number of fragments of different length with fluorescent chain terminators attached to their ends is produced. The fragments are put into a gel matrix ignited by electrical current, which sorts fragments by length. At the end of the matrix a laser reads a fluorescent label of each fragment and stores its value, thus producing the target sequence. [43] At the time of creation the technology was limited only to producing the sequences of several

hundreds base pairs a year, but over the years the technology matured leading to more automation and lower costs. Eventually the Sanger method led to the only finished-grade sequencing of human genome. Even though the Sanger sequencing dominated the field for almost 25 years, by 2000s limitations in the technology and the need for more efficient, cheaper and faster sequencing technologies became apparent.

2.1.1 Next-Generation Sequencing

In early 2000s next-generation sequencing (NGS) technologies took hold. Next-generation sequencing is an umbrella term that incorporates various sometimes radically different methods for inexpensive and efficient sequencing. A common trait of NGS technologies is massive parallelization, which is vastly superior to the maximum number of 96 sequence reads achieved with the modern Sanger technology [44]. The parallelization of the sequencing process dramatically decreases the time needed to obtain the sequence while keeping the costs low. On the other hand, Sanger method produces read lengths longer than those produced by NGS technologies. For the time being this is a clear advantage over NGS, especially with sequencing repetitive regions and *de novo* sequencing (that is determining a sequence of a previously unsequenced organism). However, the NGS technology is rapidly moving forward and the prices of sequencing are going down and NGS are expected to phase out Sanger sequencing.

Some of the most prominent NGS platforms include Roche 454 pyrosequencing, several products by Illumina, SOLiD/Life/APG and IonTorrent. Differences between these technologies are characterized by read lengths, error rate, cost, the amount of data output per run and run time. The key characteristics of these platforms, as well as the Sanger platform are summarized in Table 1 [52] [53].

Table 1: Comparison of sequencing technologies.

Platform	Read length	Error rate	Run time	Output data	Cost/Mb
Ion Torrent	200bp	1.71%	2h	<1Gb	$0.63
Roche 454	700bp	0.01%	24h	0.7b	$31
Solid/Life/APG	50–100bp	0.06%	7–14d	120Gb	$0.15
Illumina MiSeq	<150bp	0.80%	27h	1.5–2Gb	$0.5
Sanger sequencing	400–900bp	0.01%	20m–3h	1.9 84Kb	$2400

Next generation sequencing technologies include a number of methods, which generally fall into three categories template preparation, sequencing and imaging, and post-processing [45]. The technologies differ by how they perform each of these steps. The process is illustrated in Figure 2.

The template preparation begins with splitting DNA into smaller fragments. Adapters, short known DNA sequences, are then added to both ends of the fragments. Since current imaging technology does not allow detecting single events, the genetic material needs to be amplified, which is commonly achieved using a variation of PCR (Polymerase Chain Reaction). Two different types of PCR are used: emulsion PCR (emPCR) is employed by Roche 454 and Life/APG, and solid-phase amplification as used by SOLiD and Illumina. Finally the fragments are immobilized to a solid surface for support, after which they become ready for sequencing.

In the next step, the prepared template is sequenced and imaged. Illumina and IonTorrent use a sequencing by synthesis method, where new DNA fragments are synthesized out of prepared DNA templates. DNA polymerase, an enzyme that creates a DNA strand out of nucleotides, is used for synthesis by Illumina, Roche 454 and IonTorrent. SOLiD, on the other hand, employs a method called sequence by ligation, which involves another enzyme, DNA ligase for synthesis. As single nucleotides are incorporated into growing DNA strands, they are imaged and recorded as a sequence. Illumina achieves imaging by detecting fluorescently labeled nucleotides in the growing DNA strands. Roche 454 uses a chemical called luciferase to tag individual nucleotides. Luciferase emits light that can be detected for determining a corresponding nucleotide. IonTorrent, on the other hand, does not employ optical imaging methods, but instead detects pH changes, which are caused by release of a hydrogen ion during the synthesis of a DNA strand.

After sequencing is complete, raw sequence data is processed and analyzed. The first step is to filter out low quality reads and remove adapter sequences that were added in the sequencing step. The second step involves reconstructing fragments to a complete genome. In case of *de novo* sequencing the data is assembled into a complete sequence. Otherwise fragments are mapped to a reference genome. After this stage, the sequencing is complete and the reconstructed sequence is ready for further analysis and applications.

Ready availability of genetic data has dramatically changed the way biological research is conducted [43]. Instead of being a scientific curiosity, sequenced genetic data is now being ubiquitously used as an aide in life sciences research. Practical

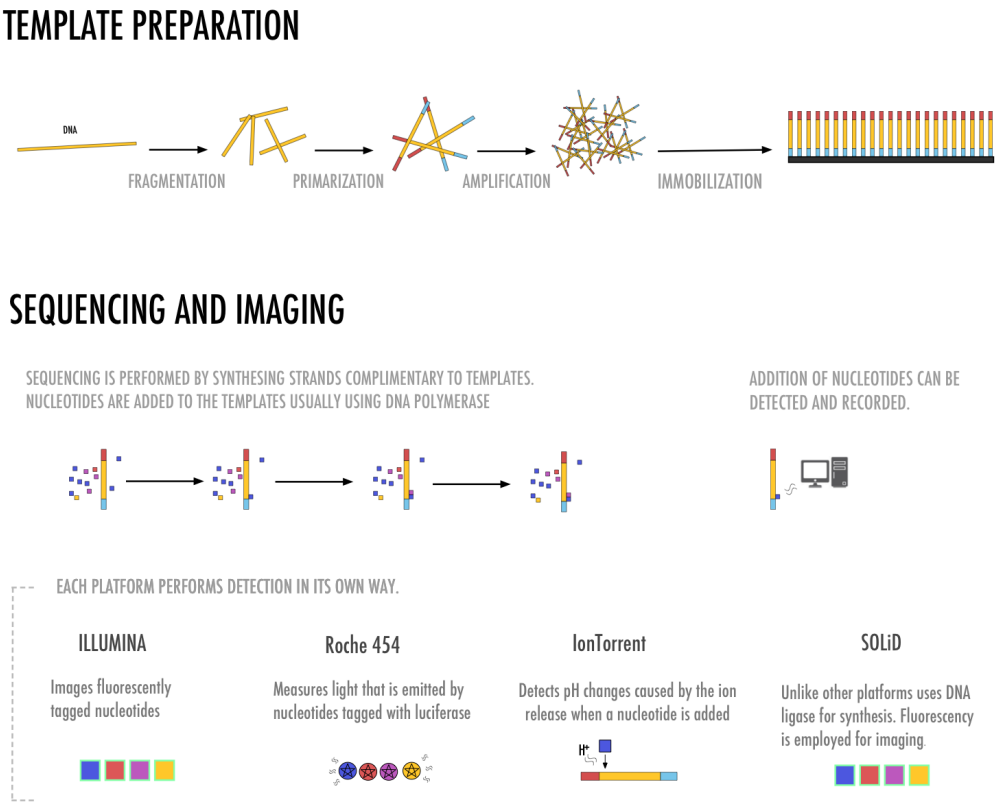

Figure 2: Overview of the next generation sequencing technology

applications of sequenced data are numerous and include among others identifying genes that result in decease processes, comparative biological studies of different organisms and study of bacterial and viral species for better understanding the underlying mechanisms [50].

2.2 RNA Sequencing

According to the central dogma of molecular biology, DNA is converted to RNA, which in turn makes proteins (Figure 3). The process of converting DNA into RNA is called transcription. The first step of transcription is a creating a copy of a DNA strand using an enzyme called RNA polymerase II. The newly created copy is called pre-mRNA (a messenger RNA) and is essentially a complementary base-paired copy of the DNA template with thymine (T) bases replaced uracil (U) bases. The resulting pre-mRNA is further transformed into the mature mRNA. The process involves RNA splicing, a process that filters out the pre-mRNA code for the parts that will be included in the final mRNA (exons), while leaving out other parts (introns). Moreover, splicing may take an alternative approach resulting in a different, but yet functional mRNA [42]. Alternative splicing might explain the phenotypical diversity of organisms, that have relatively few genes, such as humans. It is a widespread phenomenon and for example, the work of Wang [41] suggests that 92-94% of human genes are a subject to alternative splicing.

RNA sequencing (RNA-Seq) is a method to reveal a snapshot of a transcriptome using data obtained by the means of next-generation sequencing. It is similar to DNA sequencing, but unlike a genome, transcriptome data is dynamic and constantly changes based on environmental factors. Ultimately RNA-Seq allows complete annotation of all the genes and their isoforms. Even though the technology is relatively new, a number of methods to perform RNA equencing already exist. [24]

Obtaining RNA-Seq data involves several steps, which are depicted in Figure 4. The starting point is a set of mRNA that are extracted from the biological material. After the extraction, mRNAs are fragmented and converted into a cDNA library with adapters attached to one or both ends of each fragment. cDNA stands for complementary DNA and represents a fragment which is complementary to the original one according to the base pair ruling (A is complementary to T and C is to G). Once fragments are prepared they are sequenced using one of next-generation sequencing techniques. Depending on a particular sequencing method, a fragment can be sequenced either from one end (single-end) or both ends (paired-end). In the

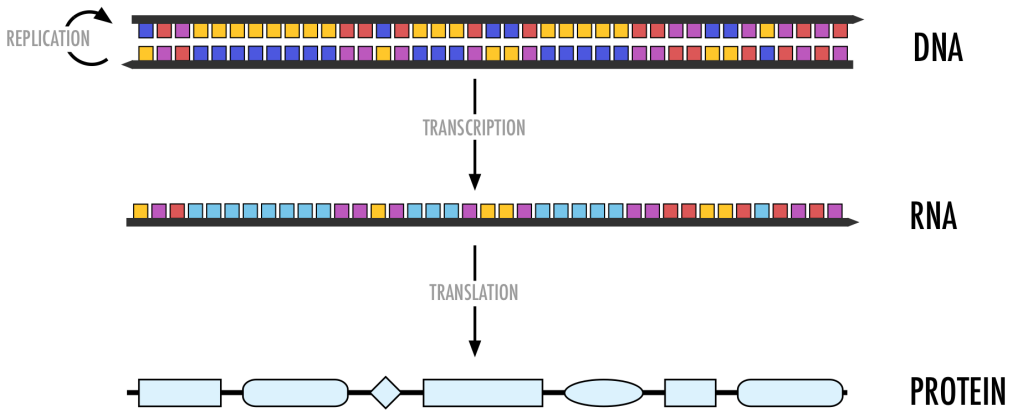

Figure 3: Central dogma of molecular biology. DNA is transcribed to RNA, which is translated to proteins.

paired-end sequencing a sequenced fragment becomes known on its ends, while the middle part remains unknown. Any next-generation sequencing technology can be used for RNA sequencing and sequencing technologies like Illumina, SOLiD and 454 Life Science have been successfully employed. [19]

One feature of these sequencing methods is a short length of reads that varies from 50 to 700 basepairs. To reconstruct the whole transcriptome the reads must be reassembled. Reassembly is not a trivial task due the presence of sequencing errors and introns in the reference genome. There are several ways to approach reconstruction. First, if a reference genome exists for the sequenced data, individual reads can be aligned to the reference data. This method is relatively easy and computationally less intensive and thus preferred. Unfortunately reference data is not always available, which might be the case for sequencing of novel organisms. If the reference data is not available, then the transcriptome is reconstructed using *de novo* assembling strategy. In *de novo* assembly individual reads are assembled into a number of De Bruijn graphs, multidimensional directed graphs that represent overlaps between symbols [19]. Individual graphs are then assembled into one graph to merge contigs and remove redundancy. In the third approach reference and *de novo* assembling strategies can be combined to detect novel transcripts. The general approach of this hybrid strategy is first to construct a partial genome using reference assembly and

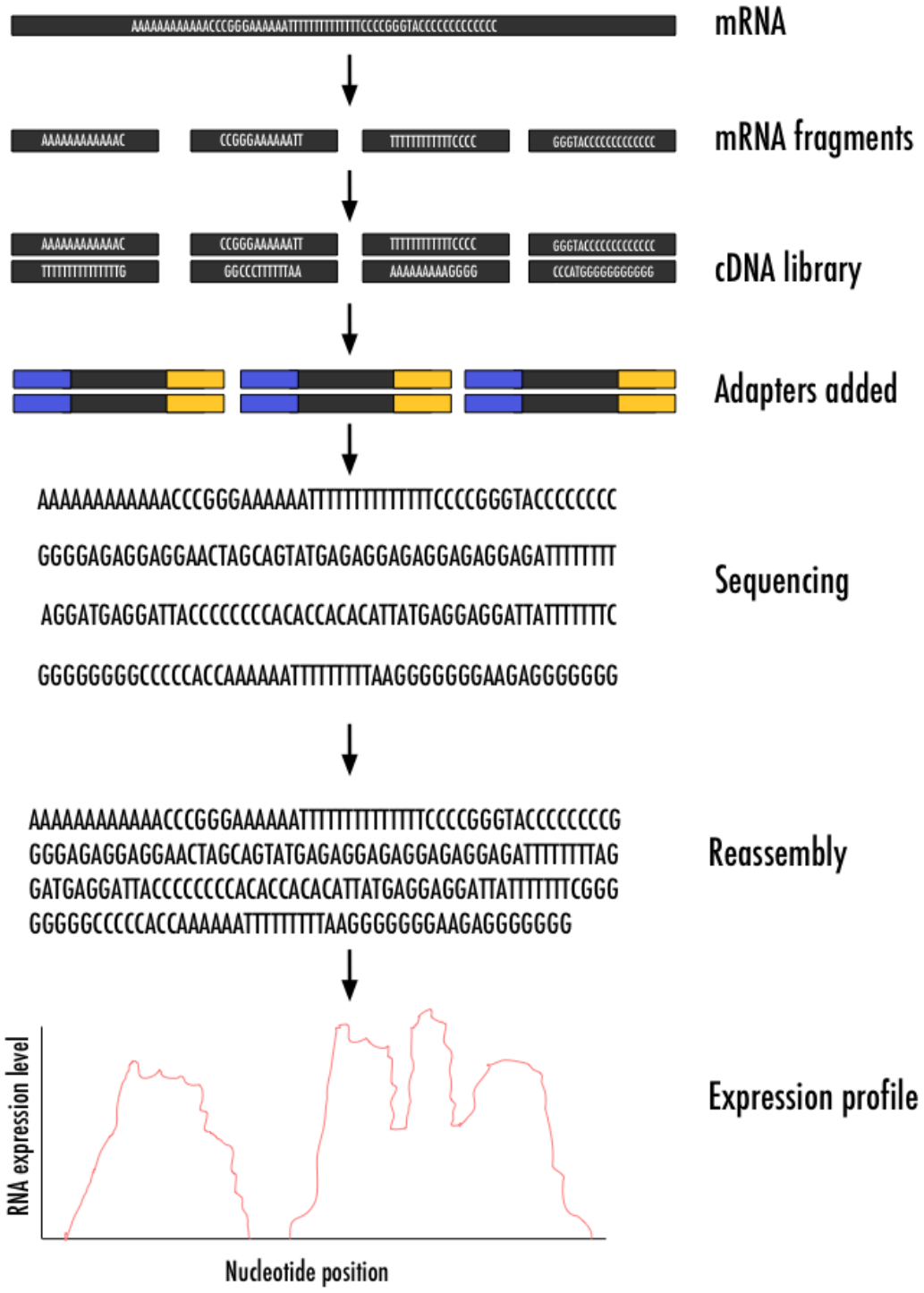

Figure 4: Overview of the RNA sequencing process

then employ a *de novo* approach. Finally after transcriptome is assembled, a gene expression profile is created and the data is analyzed by counting the number and

density of exons, splice events and new candidate genes. [19]

2.3 Anduril Workflow System

Anduril (ANalysis of Data Using Rapid Integration of aLgorithms) is an open-source cross-platform component based workflow system for scientific data analysis [14]. The system is flexible and extensible and virtually can be tailored for arbitrary data processing and analysis. However, the emphasis is made on analyzing high-throughput data produced by life science disciplines. Anduril provides out of the box support for analyzing gene expression, SNP, microarray, next-generation sequencing analysis, biodatabase mining and cell imaging analysis data among others. Adding support for analyzing data of other types is a matter of developing a new component, which can be easily plugged into the Anduril framework. Anduril has been developed and is maintained by Systems Biology Laboratory at the University of Helsinki.

The central concept in Anduril is a workflow. A workflow describes a series of processing steps that collectively perform a certain task. Each step is described by a component, a piece of reusable executable code, which accepts a number of inputs and produces one or more outputs. Steps are chained together in a pipeline with the outputs of components acting as the inputs for the next component in the pipeline. One benefit of this approach is a loose coupling, meaning that a component may be freely replaced with another one, as long as input and outputs are the same. As the bioinformatics field makes rapid advances , this is a very desirable feature that makes a quick response to technological changes possible.

Components communicate with each other using a file system. Workflows store all the output in an execution directory with a separate subdirectory for each component. The input and output of a component are represented as files in the execution directory. This approach allows workflows to save its interstate by default. If execution is halted midway, during the next run Anduril will find files from the previous run and automatically resume the workflow. Another benefit is its ability to easily analyze the output of each component separately even after the execution is finished.

Workflows are described using AndurilScript, a custom programming language. AndurilScript supports typical features found in a programming language such as variables, data types, functions, looping and branching. Components are incorporated in AndurilScript as function calls with function parameters acting as the input and a return value as the output. Furthermore, AndurilScript has a notion of annotations,

special parameters that provide additional information about a component or modifies the execution logic of component instances. For example, the @host annotation specifies the host, on which component is executed in the remote execution mode.

Below is an example of a trivial Anduril script. The script looks for input values in the files in1, in2, m1, m2. Files in1 and in2 contain simple numeric values and m1, m2 contain matrix values. Numeric values are summed, while matrix values are summed and multiplied. Finally end values are processed for output.

```
/* Numeric inputs */
in1 = INPUT(path="in1")
in2 = INPUT(path="in2")

/* Matrix inputs */
m1 = INPUT(path="m1")
m2 = INPUT(path="m2")

/* Add two integers and stores result in the variable add */
add = Add(x1=in1.in, x2=in2.in)

/* Add two matrices and stores result in the variable madd */
madd = AddMatrix(m1=m1.in, m2=m2.in)

/* Multiple two matrices and stores result in the variable mul */
mul = Multiply(x1=add.sum, x2=in2.in, x3=mstats.max)

/* Output variables */
OUTPUT(add.sum)
OUTPUT(mul.product)
OUTPUT(madd.sum)
```

2.3.1 Remote Execution

Anduril has a rudimentary support for distributed computing in the form of remote execution. Components can be executed remotely by using the @host annotation and a supplementary hosts file. The @host annotation specifies a remote host the component is intended to be executed on and the hosts file contains detailed information on remote hosts and their file systems. Provided information include server

addresses, login information, path mappings, computational slots per host and so on. Components are executed on remote hosts using SSH protocol. Public/private key based login must be set up beforehand for automatic execution. Each remote host must have Anduril and all the other required software installed. In case there is no common file system shared between hosts, required files are copied from one host to another using the *rsync* utility. Rsync is software for Unix systems that synchronizes files between different locations, both local and remote ones.

The host a workflow is run from acts as a central node, meaning that it always participates in the file copying process, acting either as the source or the destination. This approach is far from efficient and results in excessive data transfer. To illustrate the problem consider the Figure 5. Consider an example set-up involving one local node that starts an Anduril workflow consisting of two nodes each of which is executed on remote nodes. The first component is executed on the first remote node and the second component on the second node. The local node does not execute any components, but only acts as a workflow initiator and executor. Thus the local node initiates the workflow and submits a component for execution to the first remote node. Once execution finishes, next component in the workflow is executed on the second remote node. With no shared storage present data from the first execution must be copied back to the local node and then copied further to the second remote node. Even though local node does not require output data from the first component, it still acts as an intermediate transfer point resulting in excessive data transfer. On the other hand, with shared storage all the data is stored there and data is accessed only on a need basis. Furthermore, shared storage can be accessed as a local file system, which eliminates the need to copy files before accessing, further improving performance. [21]

Another problem with the remote execution is a laborious set up. Setting up remote execution implies the knowledge of remote nodes, their file systems and relevant paths. All this information must be gathered beforehand and the burden is laid on the end-user. The system is far from transparent and requires familiarity with non-trivial concepts such as setting up and using SSH keys in order to be taken into use.

2.4 Parallel Computing

Parallelization is a concept that is not only limited to computing, but it may take different forms. For example, a trench digging done manually by a single person can

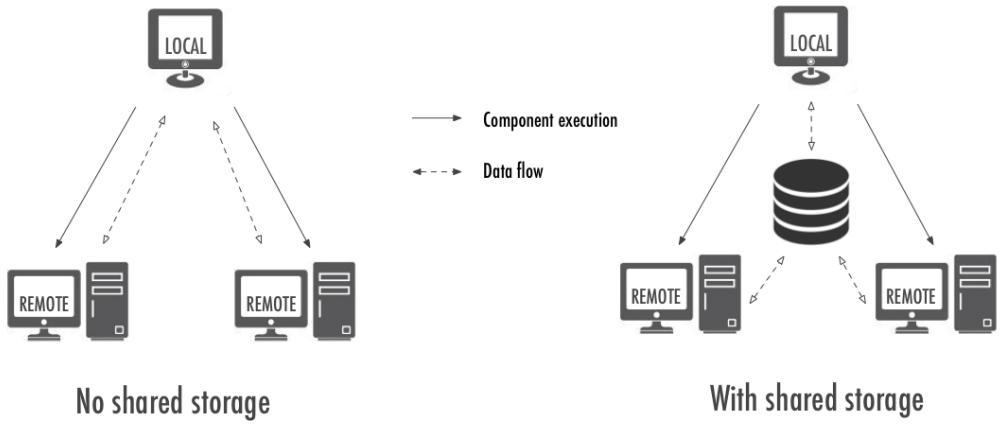

Figure 5: Data transfer when no shared storage is present and with shared storage

be sped up by employing more workers to do the job. In a perfect case the time require to finish digging the trench will be inversely proportional to the number of additional workers. This is not always achievable, as additional people might interfere with each other's work. Of course, not any kind of work can benefit from parallelization. One example of work that would not benefit from parallelization is giving a birth to a child. If it takes for a mother roughly nine months to give a birth to a child, adding more mothers the mix would not provide any benefit at all. Another type of parallelization already reviewed here is parallelization in DNA sequencing. Next generation sequencing technologies dramatically speed up a sequencing process by massively parallelizing reads. Instead of sequencing a single read at a single time, a modern NGS platform can process millions of reads at the same time. Similarly data processing can be sped up by distributing work load to several computers. As with the human example, not any kind of computing can be sped up by parallelization. For example, tasks that depend on the outcome of other tasks cannot be parallelized and must be executed in a serial fashion. On the other hand, if tasks are similar to each other, but at the same time are independent of the outcome of each other and do not require coordination between them, they can be executed in parallel. The class of the latter problems is called embarrassingly parallel [46]. Finally a third class of problems is a mixture of the first two. While a program flow is serial, it may contain parts that can be parallelized.

The RNA-Seq workflow falls under the third category. The input data consists of similar data sets that can be analyzed concurrently. There is no inter-dependencies between data sets that would prevent parallel execution. Each data set provides

parallelization to some degree, but much of it is executed in serial. So to recap different data sets are embarrassingly parallelizable, while individual data sets are a mixture of serial and parallel execution. More information on the RNA-Seq data set can be found in Section 4.2.

2.4.1 Amdahl's Law

The speed-up of a program executed in a parallel fashion is limited by the factor of how much time the program spends in a serial code. One way to measure the performance of a parallel computing solution is Amdahl's Law, which indicates the maximum speed-up of the overall system that can be achieved, when a part of the system is improved [51]. More formally the law is defined as follows.

$$S(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

$N \in \mathbb{N}$, the number of execution threads
$P \in [0, 1]$, the proportion of code that can be parallelized

For example, if half of the program code must be executed in serial, the maximum speed-up is $1/(1-0.5) = 2$, as N approaches the infinity. As the number of processors increases the P/N ratio approaches zero, which effectively sets an upper limit on the number of processors in use. For this reason parallelization makes sense for problems with a high value of P (embarrassingly parallel) or alternatively the problems with a small number of processors. As there is a hard limit on the usefulness of adding more processors, another way to improve performance is to optimize programs to make the $1 - P$ value as small as possible.

2.5 Distributed Systems

In the traditional paradigm of personal computers, computing is limited to a single computer. The computer may have several processors and each processor may have several cores, which allows to execute several computational tasks simultaneously. While this approach greatly enhances a computational potential, it is still limited by CPU and memory constraints of a single machine. To overcome these constraints computers can be connected together in a network forming a distributed system. A more formal definition of a distributed system is a collection of independent

computers that appear to its users as a single coherent system. A subclass of a distributed system is a cluster that is deployed over a local network and is tailored for a specific purpose.

One important feature of a distributed system is that the technical details and peculiarities of individual nodes and inter-node communication are hidden from the end-user. The whole system appears coherent, uniform and transparent. Ideally a distributed system would not appear any different from a single computer to the end-user. Another important feature is an ability to scale and change parts of a distributed system without affecting its functionality. Ideally computers in a distributed system can be added and removed in an ad-hoc manner without interrupting normal functioning of the system. [3]

A distributed system offers a number of benefits over traditional personal computers. First of all, it makes possible access to remote resources. Instead of equipping each computer with a large file system, a distributed file system might be used, thus simplifying access of files and resulting in economic savings. Secondly, a distributed computing power allows computing problems that are too demanding for tackling on a single computer. These problems are solved by assigning each computer with solving a part problem and finally combining results together. Thirdly, a properly designed distributed system makes possible scaling and increasing a computational potential. While it is feasible to add more computing power to a personal computer, it works only up to a certain point. There is a limit how much memory can be installed or how fast processor a personal computer is able to support. However, increasing computational potential of a distributed system is a matter of adding more nodes to the system. Ideally this process is transparent to users and can be done on the fly, even during an execution of a parallel application.

In the heart of a distributed system lies a resource manager (also known as a workload manager), a computer application that monitors and controls allocation of limited resources between jobs. A closely related concept is a job scheduler, which makes decisions on the order of execution of submitted jobs. Often these two applications are combined into one. Another essential component in a distributed system is a storage system accessible to all the nodes. A distributed storage allows to share data between the nodes without having to resort to copy files from one node to another. Below is a review of several resource managers considered in this project. Distributed file systems are reviewed in Section 2.6.

2.5.1 Evaluation criteria

The following criteria divided into three three priority categories were adopted in the evaluation of distributed solutions.

Must have

- distributed execution of jobs
- supports a shared file system between compute nodes
- consumable CPU slots to limit CPU overloading
- can be integrated into an open source program (BSD/GPL license)

High-priority

- job scheduling and load balancing
- file transfer between compute nodes without a shared file system to reduce unnecessary I/O
- pre/post-execution hooks to customize job execution (e.g., file transfer)
- mapping of file paths between file systems

Low-priority

- consumable resources to implement memory usage limiting
- easily integrated into a Java program (e.g., Java API)

More informally the goal of this work is to build a distributed system capable of sharing execution jobs between computer nodes, as well as provide a distributed storage mechanism. The solution shall work both on its own, as well as in conjunction with the Anduril software. The latter requirement applies both on a technical and a license level. High-priority features such as file transfer without a shared file system, execution hooks and mapping of file paths are designed to mirror existing features of Anduril's remote execution mechanism. While these are desired to have, they are not essential, if a shared file system is one of pre-requisites.

2.5.2 Slurm

Slurm (Simple Linux Utility for Resource Management) is an open source resource manager for Linux systems that is scalable to clusters of all sizes. Slurm is capable of managing consumable resources such as CPUs, GPUs (Graphic Processing Units) and memory among submitted jobs. Slurm is extremely scalable and is able to operate both on mini clusters of a few nodes and on gigantic clusters with tens of millions of processor cores. Slurm allows up to 1000 job submissions per second and 500 job executions per second. Furthermore, Slurm supports different scheduling policies and provides a flexible Quality of Service (QOS), which allows modifying the job queue on the fly. Slurm is fault tolerant and is capable of recovering both node and controller failures. Additionally Slurm implements a plug-in framework, which allows extending functionality beyond built-in features. Currently available plug-ins offer a wide range of additional features such as storing and viewing historical data, gathering energy consumption data, managing generic resources like GPUs among others. [4]

The architecture of Slurm consists of a single centralized server, a number of compute servers and optionally a backup server that assumes the role of the central server in the case of a failure. The central server monitors and assigns resources among jobs, as well as maintains a queue of submitted jobs and makes sure that each job get its fare share of resources according to the selected policy. On the other hand, the task of compute servers is to accept, execute, and report status of jobs.

As of June 2013, Slurm is used on half of the supercomputers in the Top500 chart. Example installations of Slurm include Tianhe-2 cluster in China (16000 nodes and a total of 3.1 million cores) and Sequoia cluster in Lawrence Livermore National Laboratory, USA (98304 nodes and 1.6 million cores). [5]

2.5.3 TORQUE Resource Manager

TORQUE (Terascale Open-Source Resource and QUEue Manager) is another open-source resource manager for Unix-like systems maintained by Adaptive Computing. It is based on Portable Batch System (PBS), a job scheduler originally developed for NASA in 1995, and features many improvements over the original PBS. It scales to clusters of tens of thousands nodes, features advanced fault tolerance, is capable of managing different types of resources and provides a more user-friendly logging facility than the original OpenPBS. TORQUE implements a client-server architecture

similar to Slurm with one head node acting as a server and many compute nodes acting as clients. [30]

2.5.4 Oracle Grid Engine

Oracle Grid Engine (OGE), formerly known as Sun Grid Engine (SGE), is an open source distributed resource management system developed by Sun Microsystems and currently maintained by Oracle. Like other workload managers described here, features of OGE include high scalability, advanced scheduling, increased reliability and flexibility. The architecture of OGE is similar to Slurm and TORQUE. Nodes in OGE are divided into master and execution hosts. The master host handles job submissions from users, assigns resources, dispatches jobs to execution hosts and monitoring the overall health status of the cluster. Additionally to increase fault tolerance, there might be a shadow master host present that assumes the responsibilities of the master host in case the master goes offline. Execution hosts accept jobs from the master host and runs them locally. Execution hosts also continuously report the status of running jobs to the master host. [17]

Even though the software started as open source, it was commercialized later and the current version features only a trial available for free. Nonetheless, there is a free fork of OGE available, Open Grid Scheduler. It is based on the original SGE and has been forked, when Sun was purchased by Oracle in 2010. The software is maintained by the original external developers of SGE, who have been contributing code since 2001. [18] However there have been no updates of software since 2011 and the project appears abandoned. The latest version of Open Grid Scheduler was evaluated, but the evaluation failed at the installation step due difficulties with compilation and lack of documentation.

2.5.5 Hadoop

Apache Hadoop [9] was also extensively evaluated for the purposes of the project. Unlike other resource managers, Hadoop is a complete computing platform for processing large data sets on clusters of commodity hardware. In addition to a resource manager Hadoop provides its own distributed file system Hadoop Distributed File System (HDFS) and a large set of related tools and technologies such as distributed databases (Cassandra, HBase), own query language (Hive), monitoring and coordination service (ZooKeeper) among others. [16] Hadoop used by thousands of

organization all over the world, including Amazon, Facebook, IBM, Twitter and Yahoo! [11].

Another striking difference is that Hadoop is not a general job execution mechanism, but a framework designed for solving problems using a MapReduce computational paradigm. MapReduce consists of two functions, map and reduce. *Map* function transforms input data into a set of intermediate key/value pairs, and *reduce* function combines all the intermediate values associated with the same intermediate key. [7] While MapReduce might sound trivial, in fact many real world problems can be solved using this algorithm. One feature of MapReduce is that any program implementing it is parallelized by default and can be executed on a cluster for high throughput performance.

2.5.6 Evaluation

With an exception of Hadoop, all the resource managers evaluated here are similar in features and meet our evaluation criteria. In particular Slurm, TORQUE and OGE fulfill P1, P3 and most of P2, except file related features. Indeed, file transfer between nodes and file path mapping, features found in Anduril remote execution mechanism, cannot be found in any of the evaluated resource managers. Instead all the resource managers rely on an underlying shared file system, which is not provided out of the box with any schedulers here with a notable exception of Hadoop.

Hadoop differed from the evaluated resource managers. Namely Hadoop offers a rather poor support for non-MapReduce means of computation. As of the Hadoop 1.x branch MapReduce is the only computational paradigm supported by Hadoop. Hadoop 2.x aims to fix the problem with the new YARN framework, which aim is to support other modes of execution [10]. YARN was evaluated, but it turned out to be extremely hard to configure and use partly due to the poor documentation and partly due to the pre-release nature of the framework. As the result it was concluded that Hadoop did not meet the most important requirement, a distributed execution of jobs. Another weakness of Hadoop is limitations posed by HDFS. See Section 2.6.2 for more details.

The commercial nature of the software was a decisive factor against deploying OGE, even when it appears to be most feature rich and comes with a solid support platform. In the end decision had to be made between Slurm and TORQUE and the former was chosen based on its wide-spread use, extensive documentation and thriv-

ing community.

2.6 Distributed File Systems

In this chapter we briefly describe basic concepts of distributed file systems, as they are an integral part of any distributed application. A distributed file system can be defined as a file system that is hosted on a server or multiple servers that allows remote access to its resources over a network. The simplest form of a distributed file system follows a client/server architecture. In this model data is stored on a remote server and accessed by clients over network. For example, Network File System (NFS) employs this architecture. In such a distributed file system files do not have to be downloaded to a local file system in order to be used (such as in a network storage system like FTP), but can be read or written directly. In contrast with FTP in order to modify a file it has to be first downloaded to a local file system and then uploaded back to the server, once the user is done with the file.

In a more elaborated view storage and access of data can be distributed over several servers, which serves several purposes. First of all, it improves fault tolerance. If one server goes down, it does not bring whole system with it, but only a respective part. Furthermore to increase robustness we might introduce redundancy by replicating data several times over multiple servers. This way a server failure would not affect data accessibility. The second impact is increased performance. By splitting files into multiple parts stored on different servers, we may access those parts simultaneously in parallel thus improving the overall performance. Ceph and GlusterFS use this strategy for example.

In the next sections we review and evaluate several distributed file systems, namely NFS, HDFS, Mapr-FS, GlusterFS and Ceph. The primary evaluation criteria is shared access of data, decent performance and POSIX compliance. POSIX is a family of standards for maintaining compatibility between file systems. In particular, we are interested in the POSIX compliance of a file system, as it would allow access a file system in a standard manner without any designated tools.

2.6.1 NFS

NFS is a widely popular distributed file system originally unveiled by Sun Microsystems in 1984. It is a part of Request For Comment (RFC) publications, an open standard that anyone is free to implement. The current version 4.1 unveiled in 2010

introduced better performance, strong security, internationalization and most importantly a support for parallel I/O in form of the pNFS (Parallel NFS) sub-protocol. [23]

The primary goal of NFS is to let computers of different architectures and different file systems to share their file system in a unified way. NFS hides details of the local file system using a standard interface called Virtual File System (VFS). This level of abstraction frees clients from knowing implementation details of the local file system of a remote host. Instead a remote file system is made to appear indistinguishable from a local one. [22]

NFS is built on top of the client/server architecture with a central server hosting all the data. While this makes a network setup straightforward, NFS lacks features of more advanced distributed file systems, like fault tolerance or a parallel access of data. Unlike more advanced file systems, the failure of an NFS server will render data unavailable. NFS mounts are also not scalable. The size of an NFS mount is dictated by the underlying partition size. The only way to add more storage to the mount size is to change the underlying partition size either by extending it or installing a new hard drive.

2.6.2 HDFS

Hadoop Distributed File System (HDFS) is a distributed file system developed by Apache Software Foundation. As the name suggest, HDFS is an integral module of the Hadoop software stack. While HDFS can be run on its own, it is best suited for running along with other services provided by Hadoop. HDFS is built with several assumptions in mind. The first assumption is that hardware failures are commonplace. For this reason, HDFS is built to withstand and quickly recover from failing hardware. Second, HDFS is designed for processing large data sets with typical files ranging from gigabytes to terabytes in size. High seek times make HDFS impractical for small files and to minimize the performance impact, HDFS sets the default block size to 64Mb. Third, HDFS is designed for streaming, rather than random access of data. For this reason, HDFS emphasizes high throughput over quick random access of data. Furthermore HDFS implements a write-once-read-many access model typical for a Hadoop application. In other words it is assumed that files are written once and never changed after that [13]. Another feature is that HDFS does not offer POSIX compliancy out of the box, but rather relies on its own API for data access. To offer POSIX support HDFS resorts to a FUSE (Filesystem

in Usespace) Linux driver. The drawbacks of this approach is poor performance and a lack of certain features, such as file append and setting file permissions. [12]

The architecture of HDFS consists of a central NameNode and a number of DataNodes, usually one per node. NameNode is responsible for the file system namespace, coordinating file access of clients and replicating data among DataNodes. DataNodes, on the other hand, are responsible for storing actual data and providing access to it. To handle possible failures of DataNodes, data is replicated several times with the common case being three times. NameNode is capable of an automatic detection of and recovering from node failures. In such a case the failed node is excluded from the cluster and data is automatically re-replicated among other nodes to conform to the replication factor. One important feature not found in other distributed file systems is data locality. The idea behind data locality is to process data on the nodes where it is stored. This approach minimizes unnecessary internode data traffic, which resulting in faster overall execution [13].

2.6.3 MapR-FS

MapR-FS is another distributed file system released as part of the MapR distribution of Hadoop. It is designed as a drop-in replacement for HDFS and fixes many shortcomings of HDFS. First of all, MapR-FS eliminates the centralized NameNode and replaces its functionality with distributed servers called Container Location Database (CLDB). Secondly, it provides native full POSIX support out of the box. Furthermore, MapR-FS provides a possibility to mount Mapr-FS partitions as NFS mounts. Thirdly, the file system implements a full random read/write support. Finally, MapR claims to have superior performance compared to HDFS with I/O throughput being 5 to 100 times faster. Mapr-FS can be used either as part of MapR distribution of Hadoop or as a replacement of HDFS in a standard installation of Hadoop. [15]

2.6.4 GlusterFS

GlusterFS [8] is an open source distributed file system designed to run on commodity hardware. Originally designed by Gluster Inc. in 2005, it is currently maintained by Red Hat. The file system is scalable up to petabytes in size and makes it easy to add additional storage on the fly. It is able to recover from hardware failures with no manual intervention, as the failure recovery and replication of data is done au-

tomatically. Unlike other distributed file systems, GlusterFS does not use separate metadata servers, but instead metadata is stored with the file data itself. Actual data is located using a designated mapping algorith of files and servers, which consolidates all the metadata into a global namespace. Furthermore, GlusterFS provides a POSIX interface using FUSE, as well as its own Application Program Interface (API) for extended functionality.

2.6.5 Ceph

The last considered distributed file system is Ceph. Ceph is a free open-source distributed storage solution scalable to petabytes in size and designed to store objects, blocks and files. Originally developed by Sage Weil as his doctoral dissertation in 2007, it has been integrated into the Linux kernel since 2010. The first major stable version was released only in 2012 and has gone through several major releases since then. Nowadays Ceph is developed and supported by Inktank Storage. The main difference to other distributed file systems evaluated here is that Ceph offers more than just a file system. In addition to acting as a file system, Ceph can act as a block device that automatically replicates and stripes data written to it among participated nodes. Data striping is a way to improve performance of data access by storing consequential segments of a file on different nodes, so that they can be accessed in parallel. Ceph implements an object storage service that allows to read and write data at a higher level in form of objects. The file system implements a POSIX interface, making it accessible as a traditional UNIX file system.

Ceph implements all the three types of storage on top of unified reliable autonomic distributed object store (RADOS) that consists of individual storage nodes. The nodes in the system are divided into object storage devices (OSDs) and monitors. OSDs are responsible for reading and writing actual data. Monitors, on the other hand, maintain a mapping of all the data in the cluster. An unique feature of Ceph is that the cluster map is stored on all the participating nodes in the cluster, including file system clients. The map is lazily propagated among the nodes in a peer to peer fashion using incremental updates, while monitors always have the freshest copy of the map. This strategy eliminates a central point or points of the failure and makes Ceph highly fault tolerant. [6]

2.6.6 Evaluation

Out of these candidates Ceph and GlusterFS appear to be the most suitable choices with their emphasis made on scalability and high tolerance. One advantage of Ceph over GlusterFS is its native POSIX support and well-written documentation, which appears to be more thorough than GlusterFS. NFS is also a good choice for remote shared storage when advanced features like distributed storage or data redundancy are not needed. The major drawback of NFSv3, a widely deployed version of NFS, is its poor optimization of parallel data access. However NFS v4.1 aims to fix this drawback by introducing parallel I/O. HDFS has many shortcomings with a lack of proper POSIX support and write once/read many philosophy being the major showstoppers. MapR appears to be a more suitable candidate than HDFS with better performance and native POSIX support. However, both systems are aimed to be used in conjunction with Hadoop, so there is little point to use them as a standalone installation.

3 Methods

3.1 Overview

As far as Anduril workflows are concerned, distributed computation can be achieved in three different ways. The first, the most straightforward way, is a parallelization of individual workflows. This way several workflows can be executed simultaneously either on the same or several machines. Considering that Anduril supports multi-threaded execution and some components are able to spawn its own threads as well, this strategy actually provides a very basic level of parallelization. The second way is to parallelize execution of components inside a workflow. Depending on the implementation of a particular workflow, independent components can be executed at the same time. The greatest strength of this approach lies in that different components can be executed on different computers overcoming resource constraints of a single computer. Finally, we can break execution of a component into individual parts and run them in parallel. This method is employed by Hadoop. In this work we consider the first two approaches, while briefly evaluate and discuss the third one.

3.2 Computing environment

The lab, where the study was conducted in, consisted at the time of 20 members working on different projects. In addition to personal computers of each lab member, there is a virtual machine environment used to run heavy jobs. The environment is shared by all the lab members and before the start of the project there was no robust reservation system in place. Anyone could run any jobs on any virtual machine with no guidance or supervision.

The virtual machine environment comprises of four virtual machines managed by CSC (Tieteen tietotekniikan keskus Oy). Each virtual machine has got 24 virtual cores and 88Gb of RAM. The environment has several distributed file system partitions, ranging in size from a couple of hundreds gigabytes to tens of terabytes. Most of the file systems are mounted as NFSv3 with an exception of two GlusterFS partitions. One of the GlusterFS partitions comprises of hard disks local to each participating nodes, while the other one consolidates the existing NFS shares to one GlusterFS partition.

During the time of testing the computing environment was shared with other lab

members. For this reason we tried to carry out testing only when the cluster was idle to minimize the impact of the jobs submitted by other users. This was not always possible and due this there might be some distortion in the test results.

3.3 Parallelization of workflows

Slurm was deployed as a resource manager and acted as a central part of the distributed computing solution. The first step was to deploy Slurm on the virtual machines and perform simple tests. After that testing was expanded to running real Anduril workflows. The last step was to instruct all the lab members to use Slurm for running their jobs. The last step was particularly crucial, as the ability to run jobs out of Slurm's framework was not removed and Slurm has no ability to take non-Slurm jobs into account. Jobs run outside of Slurm would take resources from everybody's else degrading the overall performance of Slurm jobs.

3.4 Slurm configuration

Slurm 2.3.2 was installed on four virtual machines with the default Slurm configuration. By default Slurm does not allow managing consumable resource such as CPUs or memory, but instead employs a linear FIFO (First In, First Out) scheduling. While this method makes a trivial scheduling possible, it prohibits a fine-grained resource management. To turn on consumable resource management SelectType option was set to select/cons_res and SelectTypeParameters to CR_Socket_Memory in Slurm's configuration. The latter parameter instructs Slurm to treat CPU and memory as consumable resources. The scheduler was set to sched/backfill, which is otherwise similar to the default FIFO scheduler, but starts low-priority jobs first, if their execution does not delay execution of higher priority jobs.

Additionally a wrapper script was introduced to gather and wrap commonly used Slurm options in one place. When launched with no parameters the script submits the job to Slurm queue and allocates four cores and 20Gb of RAM for the job. Given the lab environment, this configuration yields four jobs per machine on average.

3.5 Parallelization of components

The next logical step was to further fine-grain distributed computation and bring parallelization to the level of components. To achieve this Slurm was integrated

with Anduril, so that each individual component would be submitted to Slurm as a separate job. While Slurm provides its own batching mechanism in form of the *sbatch* command, we decided against it as it would introduce an unnecessary level of complexity on top of Anduril's own workflow system. Instead the implementation was build on top of Slurm's *srun* command, which allows to submit jobs to the Slurm queue in a simple manner.

The integration process was fairly straight-forward and did not involve dramatic changes to the architecture of Anduril. As Anduril components are stand-alone applications that can be run from command line, the solution was implemented by prefixing component command line strings with the *srun* command and optional Slurm's arguments.

3.5.1 Execution Modes

To differentiate various ways of executing components, a concept of execution modes was introduced. Available user modes are local (default), remote, slurm and prefix. Local is the default mode of operation, in which everything is run on the local machine using threads. Remote mode is a SSH invoke / rsync copy mechanism already present in Anduril before this project. As the name suggests Slurm mode instructs to use Slurm for running components. And finally prefix mode makes possible to put an arbitrary prefix in front of a component command line string. The mode is selected using the --exec-mode command line switch. For example to run Anduril using Slurm, one would execute a following command:

```
anduril run workflow.and --exec-mode slurm
```

To supply arguments to Slurm, one would use a --slurm-args command line parameter. Note that dashes in the argument strings must be escaped as percentage signs. For example to run a workflow as a low-priority job.

```
anduril run workflow.and --exec-mode slurm --slurm-args
"%nice 100"
```

The prefix mode is intended for introducing additional flexibility to execution of each component. For example, one could use the prefix mode as a generic mechanism that allows Anduril use with schedulers other than Slurm. Another use is to modify a component flow or gather additional information about component execution. In our tests, we used the prefix execution mode for gathering statistics about the run

of each component, such as execution time and CPU load. The prefix mode is taken into use by specifying --exec-mode prefix as a command line argument and the prefix is defined using the --prefix argument. For example to run an Anduril workflow with Torque, one executes a following command.

```
anduril run workflow.and --exec-mode prefix
--prefix "qrun"
```

3.5.2 Threads and Fine-Grained Resource Management

By default Anduril limits the number of threads per workflow to four. Obviously the number is too low for a workflow run in a distributed environment. As the solution, the default number of consecutive threads for Slurm execution mode was raised to 500. In case a user wishes to specify another limit, they could use the existing --threads command line option.

Since different components have different memory and CPU requirements, it would be beneficial to specify this information on a component level. To achieve that two new annotations, @cpu and @memory, were introduced to allow workflow developers to specify processor core and memory requirements of each component. The required memory is specified in megabytes, while the number of cores is always an integer greater than zero. The following example demonstrates how annotations are applied in a workflow file, which results in allocation of five CPU cores and 20Gb of memory for the component in question.

```
component = CSVJoin(original=in,
                    rename = "number=value",
                    @cpu=5, @memory=20480)
```

3.5.3 Load-balancing

Slurm does not provide a load balancing mechanism out of the box. The default modus operandi is to gradually fill nodes with jobs by assigning a new job to the node that has already got jobs running. Only if the available resources are not enough, the job is sent to a ”fresh” node. This strategy minimizes fragmentation of available resources and as the result maximizes available resources under a high load. On the other hand, with a low load the jobs are execute on the same node, while other nodes are idle. To achieve load-balancing, a custom shell script was introduced to

be run in the prefix execution mode. The script invokes Slurm's *squeue* command to determine a node with the least number of jobs, after which the job is sent to this node. This approach always ensure that the nodes are filled with jobs in a uniform manner.

3.5.4 Performance Evaluation

To measure performance of the distributed computing mechanism and gather statistics, workflows were run using a custom Bash prefix script. The prefix is essentially a wrapper around Slurm's *srun* command that records start and end times of the component by invoking the UNIX *time* command before running each component. The information gathered by the time command includes total running time, average CPU load and I/O throughput. Additionally the script records the name of the component and the node it was run on. All this information is written to separate files, one for each component. The data was parsed using a custom Python script that processed each file and outputted data as comma-separated values (CSV) files. The statistical analysis was performed using R software.

4 Case Study

4.1 Overview

The original case study, which our research uses as a basis, concerns a research of lymphoma, a blood cancer. The case study employs RNA-Seq technology to reveal a relationship between genes and cancer activity. More specifically the ultimate goal is to find differences in expression of certain genes associated with lymphoma in relapse and remission patients. Finding such genes may shed light on new treatment methods for this disease and lead to development of new drugs affecting discovered genes [49]. The goal of our work, on the other hand, is to speed up the RNA-Seq process by the means of parallel computing.

Cancer is an umbrella term for a variety of diseases that may take place in any body organ and that share common traits like uncontrolled and unscheduled cell proliferation. Cancerous cells fail to limit their growth and resist cell death, which can result in invade adjoining cells as well as spreading to other organs [47]. This phenomenon is called a metastasis and is the major cause of death. Cancer is amongst the leading causes of death worldwide. In 2012 alone there were 14 millions of cancer cases, which resulted in 8.2 million deaths worldwide. In two decades the number of cancer cases is expected to rise to 22 millions [48].

Lymphoma is a type of cancer that occurs when lymphocytes (white blood cells) lose their ability to regulate their growth and start to divide uncontrollably eventually forming a tumor. Lymphoma usually develops in lymph nodes, organs of the immune system found all over a body, or other parts of the body such as spleen, bone marrow or blood. Lymphoma is classified by the cell type it originates in: B, T or NK (Natural Killer) lymphocytes. The samples used as the dataset come from B-type cells, which is the most common type of lymphoma affecting 40-50% of all lymphoma cases [20].

4.2 Data set

The data set was obtained from Cancer Institute Cancer Genome Characterization Initiative (CGCI). The set comprises of RNA-Seq data of 92 diffuse large B-cell lymphoma (DLBCL) cases, divided into relapse and remission categories.

Data of each case consists of a number of paired end reads represented as FASTQ

files. FASTQ is a text-based format that is designed for storing both genetic sequences and their quality scores. The number of files per case varies from 4 to 16 divided between paired reads and their mates. Each file in the case ranges in size from 2Gb to 20Gb. The total size of the set is 9.6 Tb (Remission: 3.2 Tb, Relapse: 6.4 Tb) and there are 447 individual files in total, which we refer from now on as samples. The division of the case data into samples is dictated by how the original sequencing was done in the first place. Technically the files that make up a case could be merged together, but it would greatly reduce the parallelization potential, as less data could be analyzed in parallel. Similarly dividing files further into parts would make execution more parallel. In this study we opted to leave the files intact. The issues of splitting files and parallellization is discussed in details in Section 6.

4.3 Workflow description

The main aim of the original analysis is to clean up raw RNA-Seq data and transform it into gene expression matrices. The analysis is summarized in Figure 6. The process is divided into three stages: quality control, processing and statistical analysis.

In the first stage, quality statistics of the data is gathered using the FastQC software [25]. Low quality reads are discarded if the Phred quality score is below a threshold value of 20 for that sample, which corresponds to the 99% accuracy rate that a nucleotide is error free. The overall quality for a sample is evaluated by the number of reads that has the quality score below the mean quality score. If the number of low quality reads is above 70% (or any other user-specified value), then the whole file is discarded. Once reads a filtered, adapters/tags are removed from the remaining reads using the TagCleaner software [26]. The sequences of the adapters are not always known, as in the case when sequencing is done off-site and the information about adaptor sequences is not available. On the other hand, adapters can be predicted by the software. After the adapter removal, the reads are trimmed even further employing the Trimmomatic software [31]. The reads not meeting the minimum size criterion are discarded. Furthermore the quality at the beginning and end of reads is usually sub-par, so those must be trimmed as well. If a read becomes too short (less than 20 nucleotides), then the read is discarded. For paired end reads, the read must be removed from the both pairs. Finally FastQC is run again to check the quality of the data after the trimming and files with too few reads are discarded. This concludes the quality control stage.

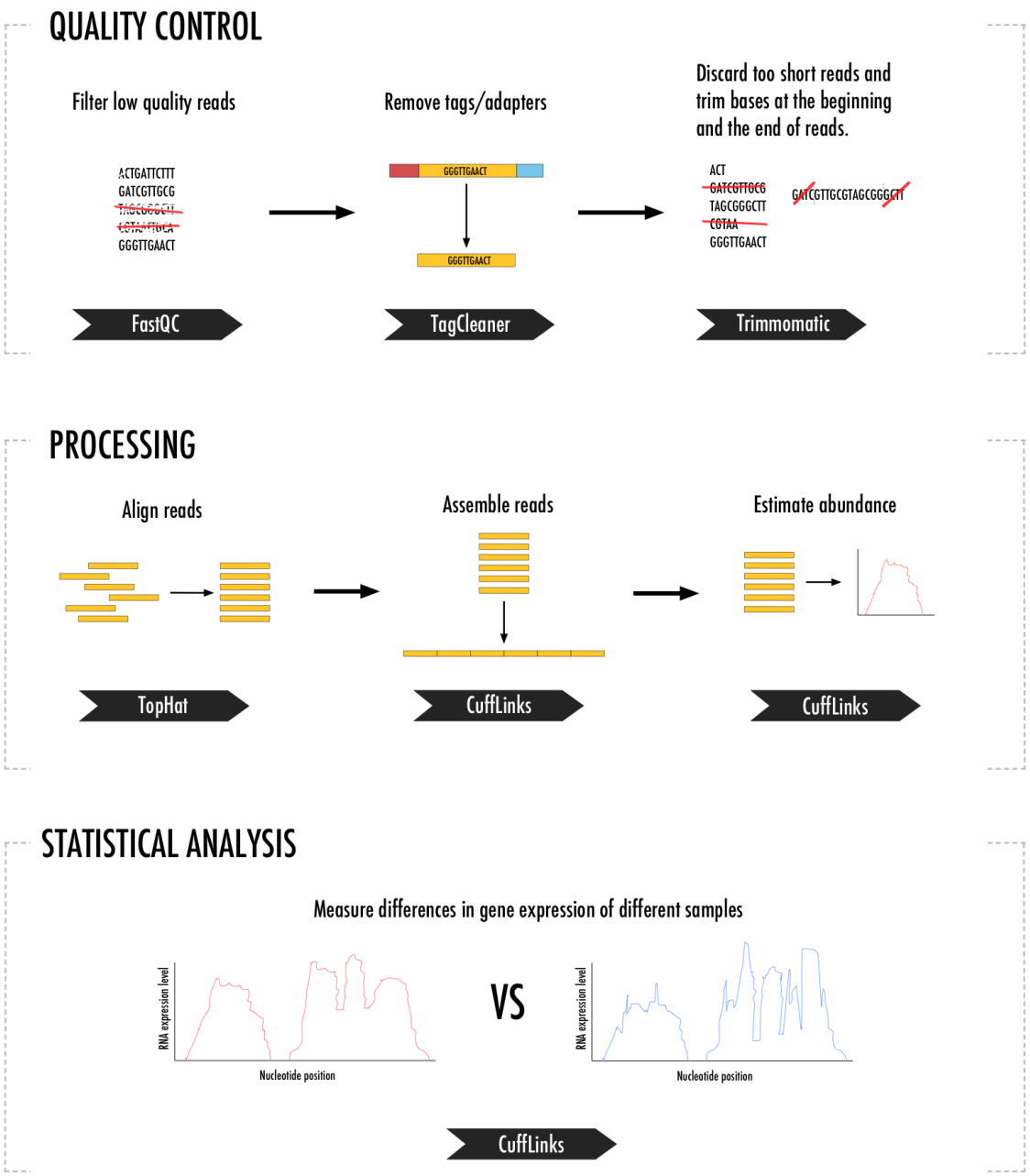

Figure 6: Overview of the RNA-Seq workflow.

In the next stage, trimmed reads are aligned and assembled into reconstructed transcripts and abundance or transcripts is estimated. Reads are aligned by running the TopHat software [27], which employs Bowtie, a Burrows-Wheeler transform aligner. TopHat is suitable both for aligning reads for which the transcriptome is available, as well as for the reads without reference genome. Another possible candidate for alignment is STAR (Spliced Transcripts Alignment to a Reference) software, which

utilizes suffix arrays and mappable seed search, clustering and stitching techniques [29]. Assembly is performed using Cufflinks [28]. In the case of the *de novo* assembly prior alignment is not required. The final step in this stage is abundance estimation, in which abundances of individual sequences are calculated and results are put into isoform expression tables. Abundance estimation is performed by CuffLinks as well. This step is important to the studies, where no control samples are available.

The final stage involves analyzing the data that has been filtered, aligned and assembled. This step involves measuring the differences in gene expression in the relapse and remission samples. Cuffdiff, a program that is part of the Cufflinks software bundle, is employed for this task.

Due to very large data sets and heavy computations, the workflow is both CPU and I/O bound. While this makes testing more complicated, it is a perfect candidate for our purposes, as it covers both types of bottlenecks. In our tests we concentrated on tuning the first stage of the workflow. While the later stages of the workflow are parallelizable, they would require a separate investigation in order to fine-tune the performance.

5 Results

We evaluated the performance of the presented solution using several metrics. The first metric is the level of parallelization. That is how many workflow components can be run simultaneously. By default Anduril allows four simultaneous components. When run on a single computer increasing the number of threads does not necessarily result in higher performance, as performance is ultimately limited by the number of cores the computer has. This constraint is alleviated in a distributed computing environment with every new node increasing the computational potential of the system. So the first metric determines whether the workflow achieves an adequate level of parallelization that would exceed capabilities of a single node. The second metric is a total execution time of the workflow. Naturally we aim for faster execution time. Finally, the third metric is CPU load of each component. It is desirable to have components to utilize the entire computational potential or in other words to have CPU load close to 100% or even greater for multi-threaded components. Lower than 100% CPU load might be a sign of I/O bottleneck introduced by delays in network communication or slow disk throughput.

5.1 Parallelization

The workflow concerns processing data sets that are perfectly independent from each other, so the problem falls under the embarrassingly parallelizable category. As there are no parts of the program that would be executed in serial, Amdahl's

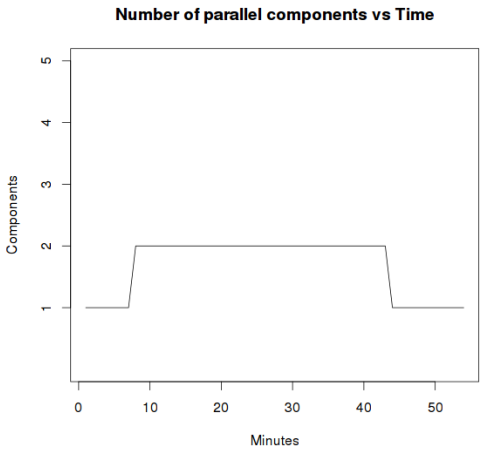

Figure 7: One sample: a number of threads over time

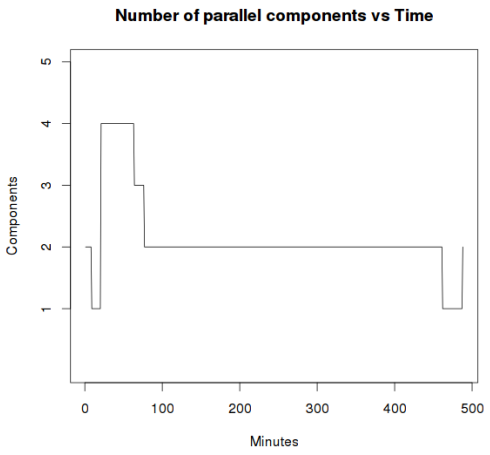

Figure 8: Two samples: a number of threads over time

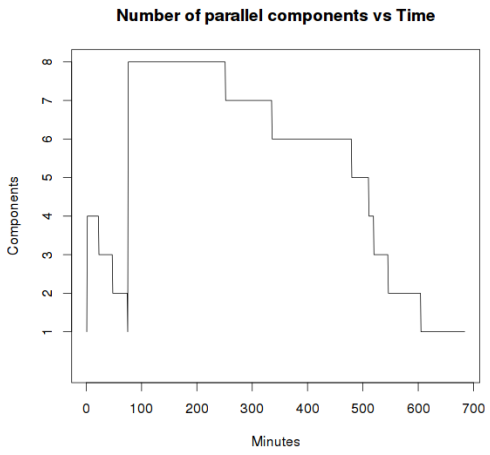

Figure 9: Four samples: a number of threads over time

law is not applicable here and the overall performance increase is limited by the limitations of infrastructure, such as throughoutput of I/O.

To make a parallelization profile of the workflow, the dataset was reduced to one sample (size 7.2Gb), after which the workflow was run on a single machine. Figure 7 depicts a number of components running in parallel over time. As we can see for the most part of the run, the workflow consisted of two parallel components. To determine whether the same condition holds with a higher number of samples we increased the number of samples first to two and then to four, after which we ran the analyses again. The situation looked a bit different this time (Figure 8) with the number of parallel components peaking at four for a period of time, but staying at two for the most part of the run. The reason for this discrepancy is a different size of each sample. The first sample is 7.2Gb and the second one is 18Gb, which is reflected in component execution times. The first sample took roughly an hour to complete, which can be seen as two additional threads running for an hour. After the analysis of the first sample was complete, the number of parallel components went back to two. The remaining two components belong to the second sample. Similar execution behavior can be observed behavior in another run of 4 samples (Figure 9). For each sample the workflow spawns two parallel component instances and the execution time of the components corresponds with the size of the sample in question.

As a next step we investigated how many threads each component of the workflow creates. To do that we employed UNIX *time* command to measure CPU load of each

component. Values above 100% indicate multi-threaded execution with a number of threads being roughly equal to CPU load divided by 100 and rounded up. For example, the CPU load of 143% indicate that the component executes two threads. We performed a test run of 12 samples and prepared a list of the longest running component along with their CPU load. Most of the longest running components turned out to be single-threaded. The AdaptorRemoval component (a wrapper around TagCleaner) which accounted for 88% of execution time is single-threaded. The SeqQC (9% total execution time), a wrapper around FastQC, has maximum CPU load of 121% (mean: 94%), but this is most likely because the program is written on Java, which incurs extra CPU load due the pecularities of Java virtual machine. Trimmomatic (4% total execution time) is clearly multi-threaded with max CPU load 200% and mean 126%, but considering its overall low impact on the total execution time, it is not worth to allocate an additional core for it. Other components have very short execution times, so can be left out of consideration altogether. To sum it up, we determined the requirements of two CPU cores per sample for the analysis workflow.

5.2 Execution time

To determine the difference in performance of the workflow run in a distributed fashion and the same workflow run on a single machine, we prepared a data set of 12 samples, which is the optimal maximum number of the samples that could be run in parallel on any single machine in our environment. More parallel components would hinder the performance of a single-machine run, as more threads than a number of cores would compete for the limited resources. On the other hand, a low number of parallel components would defeat the purpose of distributing execution over several machines.

Our assumption is that execution times of both runs would be the same. However, when run on a single machine, the workflow took 10 hours 33 minutes, but in a distributed environment it took an hour less or 9 hours 25 minutes. With such a modest dataset we shaved almost 10% off the execution time by distributing execution. The cause of overhead is not known, but the most probable cause is the hyper-threading feature of modern processors, which creates two logical cores for each physical core and share execution resources between the logical cores.

After we determined that the distributed execution did indeed provide a performance benefit, we looked at the execution times of analyses encompassing a different num-

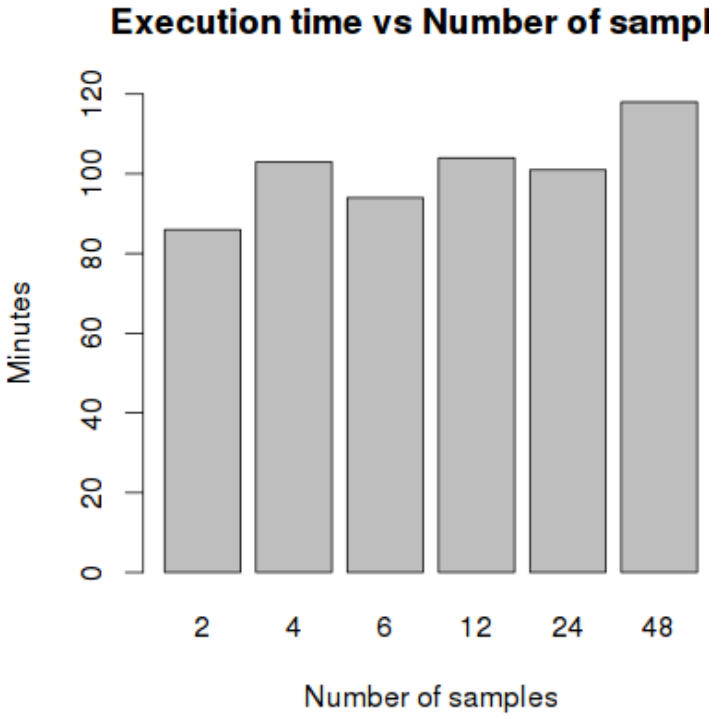

Figure 10: Execution times of different sized data sets

ber of samples. Since samples have different sizes, it makes little sense to compare execution times of individual samples to each other. In order to perform sensible testing we reduced the data set to one sample and performed analyses using this sample multiple times simultaneously. This way made possible to reliably measure execution times of different runs. Given 24 cores per machine, four machines in the cluster and two threads per sample, 48 is the maximum theoretical number of parallel components we are able to execute without impairing CPU performance. Assuming each of the analyses achieve 100% CPU load, the total execution time of these analyses should be equal to the execution time of the single sample analysis. To test this hypothesis we compared the execution times of the analyses and 1, 2, 4, 6, 12, 24 and 48 samples run in parallel. Results are shown in Figure 10.

The execution times of all the runs fall in the same range and the size of the run is not reflected in its execution time. It seems that there is a minimal overhead introduced with increased parallelization, as the the minimal run of two samples took the least time with 9% less time than the next shortest run (n=6, where n is the number of samples). Similarly the largest run of 48 samples took the most time

with 13% more time than the second longest run (n=12) and 37% more than the shortest run (n=2). Interestingly enough, execution times do not grow in a linear fashion with the increasing number of samples. This has several implications. First of all, the workflow has attended a high level of parallelization without sacrificing performance of individual components. This hints that we may further improve performance by introducing more nodes to the cluster. Secondly, there might be room for optimization here. As mentioned before, tests were performed in a shared computing environment. Indeed at times tests were run, when other jobs were present. Even though Slurm takes care of CPU and memory allocation, a similar thing cannot be said about I/O resources. We investigated this issue in the next session by measuring CPU load of each run.

5.3 CPU load

Next we analyzed the CPU load of each run. We used the data obtained from the runs in the previous section. To get comparable statistics, we calculated the mean CPU load of each component with a running time longer than 5 minutes. The reason for the cut-off is that the analysis workflow has several multi-threaded, but short-running components. Such components have no noticeable impact on the overall execution time, but would taint statistics if included. For example, in one instance the ImageGallery component had CPU load of 1015%, but ran only for 11 seconds. Similarly we opted against weighting CPU loads by the execution times, as CPU load has a direct impact on the execution time. The lower CPU load is, the slower execution performs, so a poorly performing component would have a disproportional impact on the overall statistics.

We analyzed CPU load of the analyses of 2, 4, 6, 12, 24 and 48 samples and presented results as a five-number summary and means. The results are found in Table 2. The most important observation made here is that all the runs achieved almost full CPU utilization on average, meaning that our set-up had no I/O bottlenecks. Numbers suggest that further optimizations can be made, but they would have only a minor effect on the execution times, as execution times are near their maximums already. Secondly, it appears as the number of parallel components increase, the maximum value of CPU load decreases. The cause of this behaviour is unclear. One explanation could be a bottleneck posed either by network throughout-put or by the constraints of GlusterFS, which was employed both for reading and writing data. Another possible explanation is that CPUs used in testing are hyper-threaded,

meaning that the actual number of cores is less than the nominal value and computational power is actually shared between cores. So under a heavy load, when all the advertised cores approach their maximum utilization, no additional computational power can be drawn from vacant cores.

Table 2: Basic statistics of CPU load of parallel sample analyses

N samples	Min	Q_1	Median	Q_3	Max	Mean
2	98	99	99	103	165	107
4	69	99	99	104	118	101
6	74	82	99	102	112	95
12	41	91	99	100	117	91
24	76	99	99	99	103	95
48	78	89	98	99	99	94

In another test we ran an analysis of 12 samples with a number of other jobs present in Slurm. The run took 57% more time (164 minutes vs 104 minutes) than the same run under no load. The investigation revealed that the reason for poor performance was the Trimmomatic component. All the instances of the component achieved only 9-18% CPU load. As CPU allocation is handled by Slurm, which ensures that CPUs are not overbooked, the likely culprit is either network bandwidth or file system throttling.

5.4 Heterogeneous dataset

As in previous tests the data set consisted of a single sample duplicated a number of times, it is not a valid test for real-world performance. For this reason we performed additional tests with heterogeneous samples. The newly formed data-set consisted of 24 heterogeneous samples ranging in size from 5.2Gb to 36Gb. Due the size of the data set, it would have been prohibitively expensive to run it on a single machine to establish a baseline. Instead we opted to use as a baseline the execution time of the largest sample (36Gb) and compare the total running time of all the samples to it. The hypothesis is that in an ideal situation the two running times should be along the same lines. In reality the running time of the largest sample was 179 minutes, while the entire analysis of 24 samples took 236 minutes. In other words the execution time of all the samples was 32% longer than the single sample test. Even though it is considerably longer than the single sample test, it is still a

considerable improvement over the scenario, where all the samples would be run in serial. The CPU load analysis revealed that the slowdown was caused by the SeqQC component, which is a front-end for FastQC. On average CPU load for the SeqQC component was 29%, when with one sample the CPU load was 105%. Repeated tests revealed similar results. It is not entirely clear what caused the slow-down, but the most-likely culprit is I/O. The topic is discussed in details in Discussion.

5.5 Disk space

Because of the large sizes of data sets running out of disk space was a frequent problem. Given several partitions available for data analysis free space tended to get fragmented between partitions. To illustrate the problem, consider the situation described in Table 3, which contains a snapshot of the status of mounts. The information was obtained by UNIX command *df*. Each of these mounts contain around three terabytes of space, but 13 terabytes collectively. On the other hand, there is no way to use this total free space without resorting to painstakingly moving files from one partition to another. To address the problem, a GlusterFS mount consisting of these individual partitions was introduced, which allowed to tap into this total free space. As GlusterFS allows to incorporate NFS mounts in its cluster, the solution allowed to create the GlusterFS cluster on top of existing mounts. This approach required only minimal configuration changes to the existing infrastructure and users were instructed to use this mount instead of individual ones. No additional actions were required from users.

Table 3: Snapshot of disk space usage in the lab

Partition	Size	Used	Avail	Use%
1	30T	28T	3.0T	91%
2	14T	11T	3.5T	76%
3	28T	25T	3.2T	89%
4	30T	27T	3.3T	90%

5.6 Network latency

During testing it was discovered that from time to time workflow execution failed, as components were not able to find their output files. Subsequent runs of the

same workflow did not result in the same failure, but often another failure took place after that. The culprit turned out to be latency of the NFS mount workflows were executed on. The workaround involved delaying the check of output files. The delay was implemented in a progressive fashion, starting with the delay of 50 milliseconds and doubling the delay on each subsequent check until a maximum wait time was reached (30 seconds by default, which happens to the default NFS timeout). To specify a custom maximum delay, –nfs-timeout command line option was introduced.

6 Conclusion & Discussion

We implemented and presented a distributed computing solution on top of Slurm resource manager. Testing was performed in the cluster of four virtual machines. We made modifications to the in-house workflow framework Anduril to offer better integration with Slurm and provide distributed execution of individual workflow components. Job distribution and scheduling can be used either on its own or in conjunction with Anduril. Our solution requires almost no preparation and can be taken into use with minimum prior knowledge. On the other hand, optimization of execution requires careful testing and is likely to be problem specific. For example, a number of threads created by each component is essential information that cannot be determined automatically, but must be investigated by trial.

To test the performance and the viability of the solution, the workflow for RNA sequencing data analysis was used as a case study. The performance was measured using three different metrics: the number of component executing in parallel, the CPU load of components and the total execution time. The level of parallelization achieved with the workflow was excellent and in fact far exceeded the capabilities of our testing environment. As there are 447 samples in the data set and the requirements of two CPU cores per sample, the upper level of parallelization yields 894 cores in total. In comparison our cluster is modest with only 96 cores in total. On the other hand, this paves a straightforward way to improve execution performance by adding more nodes to the cluster.

When execution times were compared, all the single sample tests took on average the similar amount of time regardless of the size of the data set. The maximum test of 48 samples took the most amount of time, but was only 13% longer than the second largest set and 37% longer than the shortest set. Comparing execution times of heterogeneous samples tests proved to be challenging because of a different size of each sample. As the result the execution time of the largest sample was chosen as a baseline and the total execution time was compared to this value. The analysis of 24 heterogeneous samples was longer than the baseline by 32%. Finally the CPU load was close to full in the simulated one sample tests. However, in the repeated test with heterogeneous samples a suboptimal CPU load was observed in several components. The possible causes for the slowdown are addressed in the Section 6.2.

A very important result, which was not covered in the results section, is solving the computer time reservation problems that had been present in the lab. Before

deploying Slurm in the lab, reservation of virtual machines was done manually. Reserving computer time involved querying lab members about their jobs and telling everyone about own jobs. The process was facilitated either verbally or by e-mail. Either way this strategy was prone to frequent miscommunication and mistakes. One problem in particular was that resource incentive jobs often crashed because of insufficient resources, when jobs by other users were ran in the same node. Needless to say this resulted in poor resource allocation and wasted both computer and human time. Slurm deployment put an end to these problems. After a short transitional period, when the lab members were instructed to use Slurm, reservation related problems vanished.

6.1 Performance improvement

As mentioned the most straight-forward, but not necessarily most cost-effective way to improve performance of the case study is to add more nodes to the cluster. Another way to improve performance is to have more samples to start from. The way the original dataset was structured, individual cases were split into separate files, from 4 to 16 files per case. This split is not by any means special, it is just a way sequencing was done in the first place. By having more files per case, we can further improve parallelization. Dividing each sample in two allows to share computation with one more extra node, reducing the time required to analyze the original sample roughly in half. On the other hand, this approach has got its practical limits for a number of reasons. First of all, we are ultimately interested in the statistics of the case, not its parts. For example while quality scores of the large samples do not differ from the score of the full case data, further decreasing the size of samples would have a noticeable effect on statistics. Secondly, a plot intended for visual inspection is generated for each sample, so each additional sample would increase the amount of manual work. Clearly with hundreds of samples it is impractical to analyze each one manually. Finally, some align software as STAR does not support multiple input files, but accept input only as a single file.

As we tested only the first out of three stages of the RNA-Seq workflow, the distributed execution strategy for the remaining two stages remains to be determined. However, it is a straightforward process, as we established necessary steps for determining resource requirements for individual components in this study. In particular the second stage of the analysis workflow, which deals with alignment, is compute-intensive and takes a lot of time. On the other hand, software used for alignment

both TopHat and STAR support multi-threaded execution. Furthermore, for both programs a number of required cores can be set as a command line parameter, so the upper bound for the alignment phase is 24 cores, which equals to the number of cores in a single node in the cluster. However, the real resource requirements could differ from the stated values, so it would be beneficial to determine that by hand. Alignment of single samples cannot be sped up further, as execution cannot be distributed to multiple nodes. However, several samples (maximum four in our cluster) can be aligned at the same time independently on different nodes.

In general the steps for determining resource requirements of a particular component are as follows. The first step is to reduce the dataset to a single entity, which can be analyzed separately and cannot be divided further in a rational fashion. In the first stage of the case study it was a single sample and in the alignment phase it is the single case data. In the second step the workflow is run with the prepared data set using a statistics prefix script or any other means that is capable of determining resource usage. After execution is finished, the statistics is ought to be analyzed manually to determine the number of parallel components and CPU load of each component. Multi-threaded components have CPU load higher than 100% and the number of required cores can be obtained by the actual CPU load divided by 100 and rounding the result up. In case there are multiple parallel multi-threaded components, the total core requirements is determined by summing up the core requirements of each component. In this project we have not run into the problem of having not enough memory, as default memory allocation settings were enough for all the performed tests. Defining memory requirements could be done using *time* command with the *-v* switch. The program reports a maximum resident set size, which is the minimum memory requirements of a component for given input data. For different sets of input data, memory requirements will be different, so it makes sense to estimate memory usage with the largest available data sample size. On the other hand, insufficient allocated memory is easy to detect, as programs tend to crash when they run out of memory, so in the best case no such estimate is needed, unless a crash occurs.

6.2 I/O issues

Sub-optimal CPU load was noticed with highly parallelized executions and especially when competing jobs from other jobs from other users were present in the system. While the reasons for the poor performance were not investigated in details, it

was determined that CPU cores were partly idle at the time, which suggests that the bottleneck is posed by I/O. The I/O system in question is composed of two components: network communication and disk throughput. The throughput of network communication is governed by the physical hardware, network topology and software configuration of network parameters of each participating node. Disk throughput, on the other hand, is defined by the disk hardware, the local file system and more importantly in our case by the type of a network file system. In our testing we employed both NFS and GlusterFS file systems. Homogeneous sample tests were carried out from a GlusterFS partition, while the data for the heterogeneous sample was stored on a NFS partition. As it was demonstrated a slowdown occurred, when heterogeneous data was analyzed. Indeed, NFSv3 file system is not designed for parallel access, which might be the culprit for the performance degradation. A simple way to test it would be to copy data to a GlusterFS partition and repeat tests. Another possible solution is upgrading to NFSv4.1, which optimizes a support for parallel access. However, the impact of a network file system on performance is beyond the scope of this project, but it would be a likely candidate for further research.

6.3 Future prospects

As biological and life science research becomes more and more dependent on computing, the need for efficient data processing will certainly grow in the future. Tools for processing data will mature and are likely to enter a more mainstream use. Similarly Anduril-like workflow processing systems that glue tools together into a coherent workflow will see wider adoption. Such a system could provide ready-made workflows for common tasks eliminating the need to implement and tweak such a workflow yourself. Amounts of data to process will grow, while the desire for shorter processing times will not go anywhere. Distributed computing is the answer that satisfies both of these requirements. While our solution is more like than a proof of concept than an end-user product, it showed that it was feasible. On the other hand it requires a fair amount of manual work for setting up and in the current form is hardly suitable for general research audience with no background in computers. Indeed, in addition to improving performance, an effort must be put into making the solution more user-friendly and easier to set up. Ideally the system would provide complete workflows for common tasks hiding the nuances of implementation and configuration details. The best scenario use case would require from the user only

to choose a correct workflow and input their data in order to get processing done.

Cloud computing is another concept that might make an impact on the life science research. In a nutshell cloud computing is a delivery of shared resources as computing power and disk space as a utility over a network [54]. As building and maintaining an own computing cluster is both expensive and complicated, it makes sense to outsource computing infrastructure to a third party. This approach eliminates the need for an end-user to get familiar with technical details, as well as paves a straightforward way for adding more processing power to computation. A possible future direction for Anduril or a similar system would involve to have a support for a cloud computing system, while acting as an interface to such a system.

6.4 Hadoop and cloud computing

We evaluated Hadoop in this project and deemed it to be unfit as a general purpose distributed computing solution. Hadoop, in its current form, is capable of only running MapReduce jobs and optimized for processing very large amount of data, which is too limiting for our purpose. Another reason was its dependence on HDFS and poor POSIX support. Given that Anduril relies a POSIX file system for its inter-component communication, integrating Anduril with Hadoop is challenging at best. Nonetheless Hadoop may be a good candidate for future endeavors, as it allows parallelize individual components beyond multithreading.

A variety of Hadoop ready tools exists for different aspects of bioinformatics computing. In particular there are tools for sequence alignment [38], mapping [33] [32], assembly [34] [35], quality assurance of sequence data [36] [37] and gene expression analysis [39] [40]. None of these tools were evaluated in details, but it appears that the current state of technology allows to carry on RNA-Seq and other kinds of bioinformatics analysis exclusively on Hadoop.

As it proved to be difficult to integrate Anduril with Hadoop given the requirements, an alternative approach could be to set up a separate Hadoop cluster and run jobs in it separately from Anduril's distribution mechanism. Alternative integration with Anduril can be carried out in form of a Hadoop bridge component that would submit a job to Hadoop from Anduril. This would allow to execute Anduril jobs normally, while compute-intensive jobs would be delegated to Hadoop. This would not, however, completely eliminate the problem of data sharing, as Hadoop requires data to be present on a HDFS partition, so data must be transferred twice.

First to the HDFS partition and after execution is complete back to the Anduril working directory. In order to alleviate this problem a HDFS partition could be used for an Anduril working directory. However, as POSIX support in HDFS is poor at the time, MapR-FS might be a better choice for this purpose given its better POSIX support and superior performance.

Another possible future solution for mitigating computational load is to perform computations in Amazon Web Services (AWS), such as Amazon EC2 and Amazon S3. Employing a third party computational power provider would eliminate the need to build and maintain a computer cluster yourself. Another benefit is hassle-free extensibility of such a solution. Amazon sells additional computer power as a product and completely shields an end-user from technical nuances. On the other hand, this would imply storing data on a third party's servers, which might pose a problem with sensitive information such as RNA-Seq data of patients' tumors. Another open question is so the technical viability of integration of Anduril with AWS remains an open question. We have not pursued this direction at all, but it might be a prospect of future research.

References

1 Gire, S. K. et al. *Genomic surveillance elucidates Ebola virus origin and transmission during the 2014 outbreak.* Science DOI: 10.1126/science.1259657. 2014.

2 Sanger, Frederick, Steven Nicklen, and Alan R. Coulson. *DNA sequencing with chain-terminating inhibitors.* Proceedings of the National Academy of Sciences 74.12 (1977): 5463-5467.

3 Tanenbaum, Andrew and van Steen, Maarten. *Distributed Systems: Principles and Paradigms.* Prentice Hall, 2nd Edition, 2007.

4 Yoo, Andy B., Morris A. Jette, and Mark Grondona, *SLURM: Simple linux utility for resource management.* Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, 2003.

5 Lawrence Livermore National Laboratory. *SLURM website.* `http://slurm.schedmd.com/` Fetched 2013/10/12.

6 Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., & Maltzahn, C. *Ceph: A scalable, high-performance distributed file system.* Proceedings of the 7th symposium on Operating systems design and implementation (pp. 307-320). USENIX Association. 2006.

7 Dean, Jeffrey and Sanjay Ghemawat. *MapReduce: simplified data processing on large clusters.*. Communications of the ACM 51.1 (pp. 107-113). 2008.

8 RedHat Inc. *Gluster Community Website.* `http://www.gluster.org/`.

9 The Apache Software Foundation. *Hadoop Official Website.* `http://hadoop.apache.org/`.

10 The Apache Software Foundation. *Apache Hadoop NextGen MapReduce (YARN).* `https://hadoop.apache.org/docs/current2/hadoop-yarn/hadoop-yarn-site/YARN.html`.

11 The Apache Software Foundation. *Hadoop Wiki: Powered by Hadoop.* `http://wiki.apache.org/hadoop/PoweredBy`. Fetched 2013/10/14.

12 The Apache Software Foundation *Hadoop Wiki: Mounting HDFS*. `http://wiki.apache.org/hadoop/MountableHDFS`. Fetched 2013/10/12.

13 *HDFS Architecture Guide*. `http://hadoop.apache.org/docs/stable/hdfs_design.html`. Fetched 2013/10/12.

14 Ovaska, Kristian. *Component-based workflow framework for gene expression microarray analysis*. MSc Thesis, 2008.

15 MapR. *MapR Distribution for Apache Hadoop Advantages*. `http://www.mapr.com/products/why-mapr`. Fetched 2013/10/16.

16 White, Tom. *Hadoop: the definitive guide.*. O'Reilly, 2012.

17 *Oracle Grid Engine: An Overview*. An Oracle White Paper, `http://www.oracle.com/technetwork/oem/host-server-mgmt/twp-gridengine-overview-167117.pdf`. 2010.

18 *Open Grid Scheduler*. `http://gridscheduler.sourceforge.net/`. Fetched 2013/10/10.

19 Martin, J. a. and Wang, Z. *Next-generation transcriptome assembly*. Nature Reviews Genetics, 12,10(2011), pages 671-82. `http://www.ncbi.nlm.nih.gov/pubmed/21897427` 2011.

20 Kumar, V., Abbas, A., Aster, J. and Robbins, S. *Robbins Basic Pathology* Chapter 11 Hematopoietic and Lymphoid Systems. Elsevier Saunders 2013

21 Kristian Ovaska, Ping Chen, Marko Laakso, Ville Rantanen, Riku Louhimo, Sirkku Karinen, Javier Núñez Fontarnau, Vladimir Rogojin, *Anduril User Guide*. 2013.

22 Sun Microsystems Inc. *RFC 1094: NFS: Network File System Protocol Specification*. Network Working Group. March 1989.

23 Shepler S., Storspeed Inc., Eisler M., Noveck D. *RFC 5661: Network File System (NFS) Version 4 Minor Version 1 Protocol*. Internet Engineering Task Force (IETF). January 2010

24 Garber, M., Grabherr, M. G., Guttman, M., & Trapnell, C. *Computational methods for transcriptome annotation and quantification using RNA-seq.*. Nature methods, 8(6), 469-477. 2011

25 Babraham Bioinformatics *FastQC: A quality control tool for high-throughput sequence data.* http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

26 Schmieder, R., Lim, Y., Rohwer, F., & Edwards, R. *TagCleaner: identification and removal of tag sequences from genomic and metagenomic datasets.* BMC bioinformatics, 11(1), 341. 2010

27 Trapnell, C., Pachter, L., & Salzberg, S. L. *TopHat: discovering splice junctions with RNA-Seq.* Bioinformatics, 25(9), 1105-1111. 2009

28 Trapnell, C., Williams, B. A., Pertea, G., Mortazavi, A., Kwan, G., van Baren, M. J., ... & Pachter, L. *Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation.* Nature biotechnology, 28(5), 511-515. 2010

29 Dobin, A., Davis, C. A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., ... & Gingeras, T. R. *STAR: ultrafast universal RNA-seq aligner.* Bioinformatics, 29(1), 15-21. 2013

30 Adaptive Computing *TORQUE Resource Manager* http://www.adaptivecomputing.com/products/open-source/torque/

31 Bolger A, Giorgi F. *Trimmomatic.* http://www.usadellab.org/cms/index.php?page=trimmomatic/

32 Nguyen, T., Shi, W., & Ruden, D. *CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping.* BMC research notes, 4(1), 171. 2011

33 Pireddu, L., Leo, S., & Zanetti, G. *SEAL: a distributed short read mapping and duplicate removal tool.* Bioinformatics, 27(15), 2159-2160. 2011

34 Schatz M, Gupta A, Gupta R, et al. *Contrail Home Page.* http://sourceforge.net/apps/mediawiki/contrail-bio/index.php?title=Contrail

35 Prasad VVS, Loshma G. *HPC-MAQ: a parallel short-read reference assembler.* CCSEA 2011 2011.

36 Kelley, D. R., Schatz, M. C., & Salzberg, S. L. *Quake: quality-aware detection and correction of sequencing errors* Genome Biol, 11(11), R116. 2010

37 Robinson, T., Killcoyne, S., Bressler, R., & Boyle, J. (2011). *SAMQA: error classification and validation of high-throughput sequenced read data.* BMC genomics, 12(1), 419. 2011

38 Matsunaga, A., Tsugawa, M., & Fortes, J. *Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications.* In eScience, 2008. eScience'08. IEEE Fourth International Conference on (pp. 222-229). IEEE. 2008

39 Langmead, B., Hansen, K. D., & Leek, J. T. *Cloud-scale RNA-sequencing differential expression analysis with Myrna.* Genome Biol, 11(8), R83. 2010

40 Hong, D., Rhie, A., Park, S. S., Lee, J., Ju, Y. S., Kim, S., ... & Seo, J. S. *FX: an RNA-Seq analysis tool on the cloud.* Bioinformatics, 28(5), 721-723. 2012

41 Wang, Eric T., et al. *Alternative isoform regulation in human tissue transcriptomes.* Nature 456.7221 (2008): 470-476.

42 Clancy, S. *RNA splicing: introns, exons and spliceosome.* Nature Education 1(1):31 2008

43 Metzker, Michael L. *Emerging technologies in DNA sequencing.* Genome research 15.12 (2005): 1767-1776.

44 Hutchison, Clyde A. *DNA sequencing: bench to bedside and beyond.* Nucleic acids research 35.18 (2007): 6227-6237.

45 Metzker, Michael L. *Sequencing technologies - the next generation.* Nature Reviews Genetics 11.1 (2009): 31-46.

46 Barney, Blaise. *Introduction to parallel computing.* Lawrence Livermore National Laboratory 6.13 (2010): 10. APA

47 Hanahan, Douglas, and Robert A. Weinberg. *The hallmarks of cancer.* cell 100.1 (2000): 57-70.

48 World Health Organization. *Cancer Fact sheet №297.* Updated February 2014 `http://www.who.int/mediacentre/factsheets/fs297/en/`

49 Cervera, Alejandra. *Computational framework for systematic and scalable analysis of deep sequencing transcriptomics data.* MSc Thesis, 2012.

50 Grada, Ayman and Weinbrecht, Kate *Next-Generation Sequencing: Methodology and Application.* Journal of Investigative Dermatology (2013) 133, e11; doi:10.1038/jid.2013.248

51 Amdahl. G.M. *Validity of the single-processor approach to achieving large scale computing capabilities.* In AFIPS Conference Proceedings, vol. 30 (Atlantic City, N.J.. Apr. 18-20). AFIPS Press, Reston. Va., 1967. pp. 483-485.

52 Liu, Lin, et al. *Comparison of next-generation sequencing systems.* BioMed Research International 2012 (2012).

53 Quail, Michael A., et al. *A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers.* BMC genomics 13.1 (2012): 341.

54 Mell, Peter and Grance, Timothy. *The NIST Definition of Cloud Computing.* National Institute of Standards and Technology `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`