A Framework to Process Large Quantities of Data Using Cloud Computing

Faculty of Science
Business Informatics

Andrew Kfoury
ID: 6893767

Supervisors:
Wienand Omta, Johan Jeuring, and Gerard Wagenaar.

Abstract

In a world where data is getting bigger and time is valuable, new needs arise to analyze big data in a timely manner while keeping costs controlled. There are certainly existing methods and frameworks to do so. However, these methods present their own drawbacks and weaknesses like compatibility or scalability. There are currently hardly any methods or frameworks that tackle cost efficiency, performance, scalability and compatibility all at once. In collaboration with Core Life Analytics on a graduation project, this thesis aims to fulfill or help accomplish that need.

Acknowledgements

I would like to first and foremost thank my internship supervisor Wienand Omta for helping me write my thesis. Wienand, I really can't thank you enough for spending countless hours teaching me about coding, statistics, networking, computer hardware, and all about AWS. I would also like to thank you for being a friend and giving me some life advice when I faced hardships throughout the last year.

Secondly, I would like to thank my supervisor Johan Jeuring for always making time for me when I asked, although he’s an extremely busy person. Thanks for everything Johan!

Third, I would like to thank my second supervisor Gerard Wagenaar for giving me valuable comments on my thesis that helped tremendously.

Fourth, I would like to thank Core Life Analytics, Especially Job and Michel, my fellow developers, for helping me, even during late night hours when I had technical issues with my code or with the experiments.

Last but not least, I would like to thank my girlfriend, for always being there for me, mentally and physically. She has always listened to my frustrations, gave me advice, and always cooked for me while I was busy writing this thesis. Thank you very much Rachel, I love you from the bottom of my heart.

Table Of Contents

Chapter 1: Introduction

Companies are swimming in an ever-expanding sea of data. That data is either too voluminous to be managed or too unstructured to be analyzed, sometimes even both (Davenport et al., 2012). In the last decade, Big Data has been at the center of attention in the IT field and is mainly characterized by 5 Vs:

- Volume is the size of the data. Volume presents the most immediate challenge to IT architectures, which is storage.
- Velocity is the speed at which data is generated or added to the existing data set. Velocity also affects the speed at which the data can be processed and analyzed by relational databases.
- Variety is the structurability and form of the data (pictures, PDFs, emails, etc.), which in itself is divided into three categories: Structured, semi-structured, and unstructured data. Each variety usually requires different capabilities and algorithms to be analyzed.
- Veracity is the accuracy of the data. When faced with high volume, velocity, and variety of the data, it might be possible that not all that data is 100% accurate or correct. Subsequently, the analysis of the data depends on the veracity of the source data.
- Value; Big Data can potentially be of great importance to companies and enterprises. However, that value becomes apparent only if the data is analyzed correctly; otherwise, it can be wholly useless and costly (Ishwarappa, 2015).

With the right tools and correct data analysis, Big Data has the potential to be a diamond when mining for gold. Big Data can provide useful insight and critical decision-making information to an enterprise or company. Big Data is present in many applications and fields like finance, biology, chemistry, economics, and business. For example, in business, big data can be analyzed to lower costs and boost profits resulting in a competitive advantage (Chen et al. 2017). Big data can similarly be a stepping stone in discovering a new cure for a disease or simply predicting the weather.

However, it is not always rainbows and unicorns when handling Big Data. Most of the data is unstructured by nature, it can come in many formats as it can be collected from mobile phones, social media, GPS signals, sensors, and many more. Handling unstructured media can prove to be a tedious task since unstructured data can’t be easily stored in the form of tables (rows and columns). Converting unstructured data to structured data can be an expensive and time-consuming task.

Incomplete and missing data is also a common problem faced with big data, which may be caused by improper recording of sensor data, skipping values while reading from a device, etc. This is sometimes problematic since it creates uncertainty when trying to analyze the data.

Additionally, due to the volume and velocity of big data, it can be costly to store and manage (S. Bagga and A. Sharma, 2014). This phenomenon has driven big names like Amazon, Microsoft, and Google to not just grasp the opportunities that big data provides, but also to come up with solutions to the problems that arose with it, like storing the vast amount of data and analyzing it.

1.1 Analyzing Big Data

Initially, Big Data analytics required a new framework and set of tools since it was not possible to analyze it in traditional ways (number crunching in tables with rows and columns). Frameworks like Hadoop and MapReduce enabled users to analyze big data by bringing processing power to the data rather than bringing the data to the processing power. However, these tools often need a powerful computing cluster and fast network connections, which also may result in heavy data traffic. Additionally, big data analytics packages like Hadoop rely on using distributed computing, however, traditional database systems do not always provide the storage scalability that big data requires (Patil & Phursule. 2014). Furthermore, more traditional packages like R, a programming language and framework made for data analysis, also benefit from distributed computing. Usually, organizations use Hadoop to handle the data volume, and then when the data is summarized and cleaned, they use traditional tools to further analyze the data (Özcan et al., 2011). This whole process has made Cloud Computing the perfect match for the job.

As stated by the US National Institute of Standards and Technology: “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” (Mell P, and Grance T. 2011).

Nowadays, all major cloud providers offer virtual machines (VMs) to suit the needs of their customers. According to the most recent press release in Q4 2020, Amazon Web Services (AWS) leads the market with 31% of the market share, followed by Microsoft (Azure) with 20%, Google with 8%, and lastly Alibaba Cloud with 7%. Similarly to its competitors, AWS offers a wide range of VMs that can be customized to fulfill an expansive list of purposes like general-purpose, computing optimized (High Clock Speed CPUs), memory-optimized (more and faster RAM), storage optimized (dedicated storage with high IOPS or a very high capacity of storage), and accelerated computing instances (instances with one or more graphical processing unit (GPUs). At AWS it is also possible to have hybrid instances that can be optimized for multiple applications, i.e compute and memory-optimized, general-purpose but with high network throughput, etc. (Bao, Damon, Lanman & Gokhale, 2016). These instances can be spun up

in 2 ways: On-Demand and Spot. With the on-demand option, an instance is reserved for the user which can be accessed at any given time as long as the user pays an hourly rate for it (based on AWS' capacity). The spot option gives the user the ability to use AWS's overhead instances at a cheaper price at the risk of being terminated when reserved by an "on-demand" user or when other AWS users bid a higher price. Not only do some instances have their own storage, but they can also be linked to Amazon's Elastic Block Store (EBS). EBS allows users to create physical storage volumes and attach them to AWS instances in various configurations optimized for high IOPS (input/output operations per second) or high throughput. Additionally, AWS instances can also query Amazon's Simple Storage Service (S3), a scalable object storage service that can be used to run everything from data lakes, websites, mobile applications, backups, archiving, big data analytics, etc. (https://aws.amazon.com/, n.d). With that being said, AWS would make the perfect environment to run big data analytics as it provides both scalable computing performance and scalable storage to store all the data at a reasonable price (if done properly).

1.2 Efficiency in data processing:

Providing virtual machines to analyze and store big data is definitely a possible solution for a big data analytics challenge. However, there are important factors to take into consideration when analyzing large volumes of data.

1.2.1 Time

At AWS, it is possible to rent in a pay-for-play fashion a rather powerful virtual machine e.g. containing 64 cores and 512 Gigabytes of RAM such as the r6g.16xlarge. This machine is quite powerful and capable of chewing through the hardest of tasks. This machine is similar to a Troll (a mythical beast, with superhuman strength but minimal intelligence), all brawns but no brains. Even with this machine, it could still take months to analyze large data sets due to their complexity and intensity. Thus, using cloud computing blindly without any knowledge nor strategy to analyze big data does not yield many benefits. For the purpose of this thesis, "efficiency" in data processing is demarcated using four main components: Granularity, Code Optimization, Right Hardware, and Parallelization (Ross, 2013).

Figure 1: Four components demarcating efficiency in this research.

1.2.2 Granularity & detail

Granularity can be described as the "resolution" of the data. In other words, how detailed does our dataset allow the data scientist to zoom in on the data at hand? Granularity can often be a double-edged sword. It can provide meaningful insights to the analysis but result in an increase in data size. While that tradeoff might sometimes be worth it, it can often be the case that the additional detail has no added value to the analysis. Usually, data scientists can get to a point of saturation or satisfaction using less data. Therefore, choosing the right resolution or granularity and avoiding ultra-high dimensionality can increase the efficiency of the data analysis a 10-fold or more (Thoman P., Jordan H., Fahringer T., 2013).

1.2.3 Hardware

Choosing the right hardware is crucial in reducing the required time span to finish a computational task while minimizing the costs. As stated earlier, AWS has these behemoths of virtualized machines that can be rented. AWS also offers hundreds of instances that are less powerful but are far more optimized for specific tasks. Renting these big machines may result in an overhead of resources that might not be used to their fullest potential.

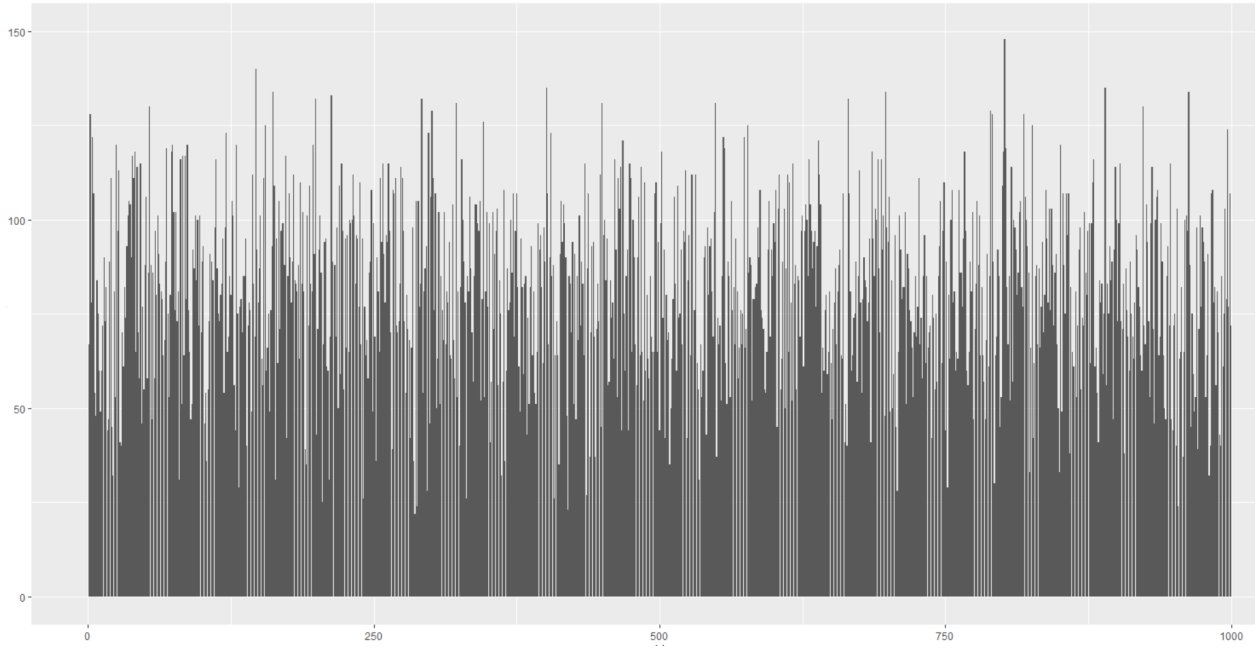


Figure 2: Hypothetical Load on server

Figure 2 shows the hypothetical load distribution even when a cloud server is under heavy load. In practice, a rented cloud instance will rarely reach its maximum load capacity even when multiple users are using it.

1.2.4 Code Optimization

Optimizing code can have dramatic improvements in speed.

For this paper, the primary programming language used will be R. Additionally, code optimization will be broken down into two main factors:

- Compiling: R is one of the go-to scripting languages for data analysis nowadays, as it is a powerful tool to manipulate and visualize data. R code is interpreted while running it, where other languages are usually first compiled before execution. Usually, R code runs slower than other languages. R however, can make use of several powerful packages like the Rcpp package. Rcpp gives R access to a C compiler, several existing C libraries, and efficient memory usage (Eddelbuttel et al., 2011). Additionally, R does have a compiling ability by calling the cmpfun() function in the base compiler package, which can speed up functions by a factor of 3 or 4 in some cases.
- Code Efficiency: There are multiple ways of writing the same functions, or in other words, there are several ways one can write code to serve the same purpose. However, not all of these ways work equally, different ways present different execution times or even slightly different

functionality. Benchmarking is a way to test and compare the speed at which functions are executed. The system.time() function and microbenchmark package in R can be used to compare execution times for functions written in different ways. For example, R can handle an object type called data frame. At the time of their creation, data frames were an extremely powerful tool that allowed faster data manipulation. However, as R grew, packages like data.table were introduced that offered exactly what data frames offer, but better (Dowle et al., 2015). The "data.table" package is a wrapper for data frames and can handle all functions that were originally designed for data frames. The package is written in C and is therefore a very efficient library containing functions using pointers to avoid copies and use the available memory extremely efficiently. Together with the library "parallel", (McCallum, E., & Weston, S., 2011), which offers multithreading functionality, "data.table" library offers a faster execution time than the data.frame way. That is just an example of how different packages can make a big difference. Nevertheless, code efficiency can introduce syntax complexity and can even make it unreadable for outsiders. Therefore, having a clear vision of what the code needs to do, the time it takes to execute, and if it will be revisited in the future will greatly impact its usefulness (Ross, 2013).

1.2.5 Parallelization

At its core, parallelization is doing multiple tasks simultaneously to save time. Parallelization is present in business process management, construction, cooking, and many fields. Parallelization within the architecture of a computer with one CPU is called multithreading. Multithreading can use all potential available CPU cores to execute multiple tasks simultaneously. Computational parallel processing applies the split-apply-combine method (Wickham, 2009), which basically breaks up a task (or data), distributes the split tasks among multiple cores, applies a certain function or calculation to each chunk of data and then consolidates all the results into one output (Wickham, 2011). Nowadays, parallel processing can be implemented in R with very little programming time. Implementing parallel processing on the average workstation or a typical laptop with multiple cores can speed up a program or processing time by a factor of 2 to 4 (Ross, 2013). This factor can be boosted by increasing the number of CPU cores. Parallelization can be executed on a single machine i.e. multithreading or multi-core, parallelization can also be applied using multiple machines (also called nodes) or multi-node. On paper, parallel processing seems to be the go-to way of doing tasks, yet parallel processing can sometimes be a doubtful advantage. Parallelization sometimes increases execution time because the amount of work to delegate is greater than the actual work that needs to be done. In other words, if the task is too small then it is not worth parallelizing it in terms of time. Parallelization works best when a workload or problem is "Embarrassingly Parallel",

meaning the task at hand requires little to no effort to be separated into a number of parallel tasks. It is often the case that communication between tasks in an embarrassingly parallel problem is not needed, thus making the problem optimal for parallelization. Parallelization presents drawbacks when the split tasks require input from another ongoing task. For example, it is very hard to parallelize a population dynamics simulation, where for each time step, the output of the previous time step is needed, resulting in time steps being inefficient to split up (Thoman P., Jordan H., Fahringer T., 2013).

1.3 High Content Screening (HCS)

Life science research has increasingly become reliant on data analysis (Leonelli, 2012). Industrializing the drug discovery process was, and still is, one of the main goals of the biochemistry and pharmaceutical industries (Williams, 2011). This goal was made possible with the introduction and application of High Throughput Screening (HTS). HTS involved the use of robotics to carry out large-scale miniaturized biological experiments in an automated fashion in standardized microplates (Mayr, & Fuerst, 2008).

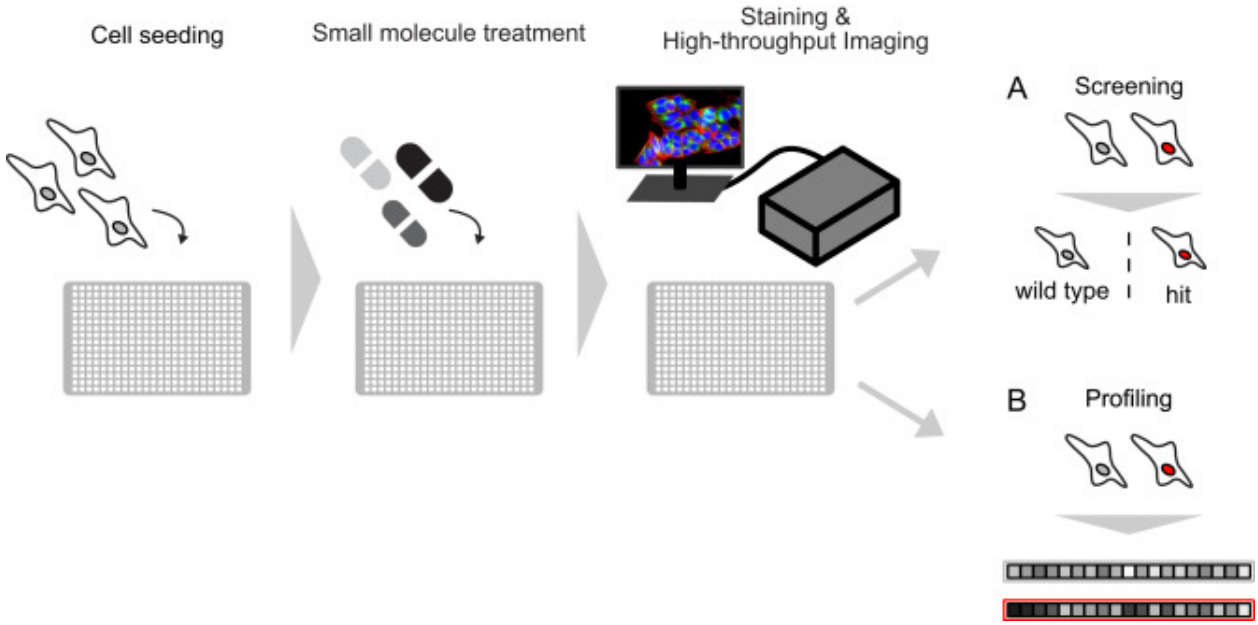


Figure 3: Simplified HCS Workflow Example

The main goal of HTS is to screen reagent libraries against biological assays using fluorescent proteins, chemical substrates, antibodies or cells. The process was conducted with minimal human interaction to maintain accuracy and consistency. Therefore, most of the drug dosing is executed by robots. HTS allows the quick discovery of active molecules, genes, or antibodies that affect relevant

biological processes. That output was essential to understand, design, and develop therapeutic drugs (Kraljevic, Stambrook & Pavelic, 2004).

High Content Screening (HCS) shares many core techniques with HTS. The major difference between the two is that HCS gets multiple readouts from cell analysis.

In HCS, live cell imaging is involved. It aims at extracting temporal dynamics and spatial information and, thus, there is a heavy dependence on image analysis. The objective of high content screening is to balance collecting comprehensive information with high efficiency. HCS typically uses 96, 384, or sometimes 1536 microplates (Figure 4) (Mcdonald P, et al., 2008).

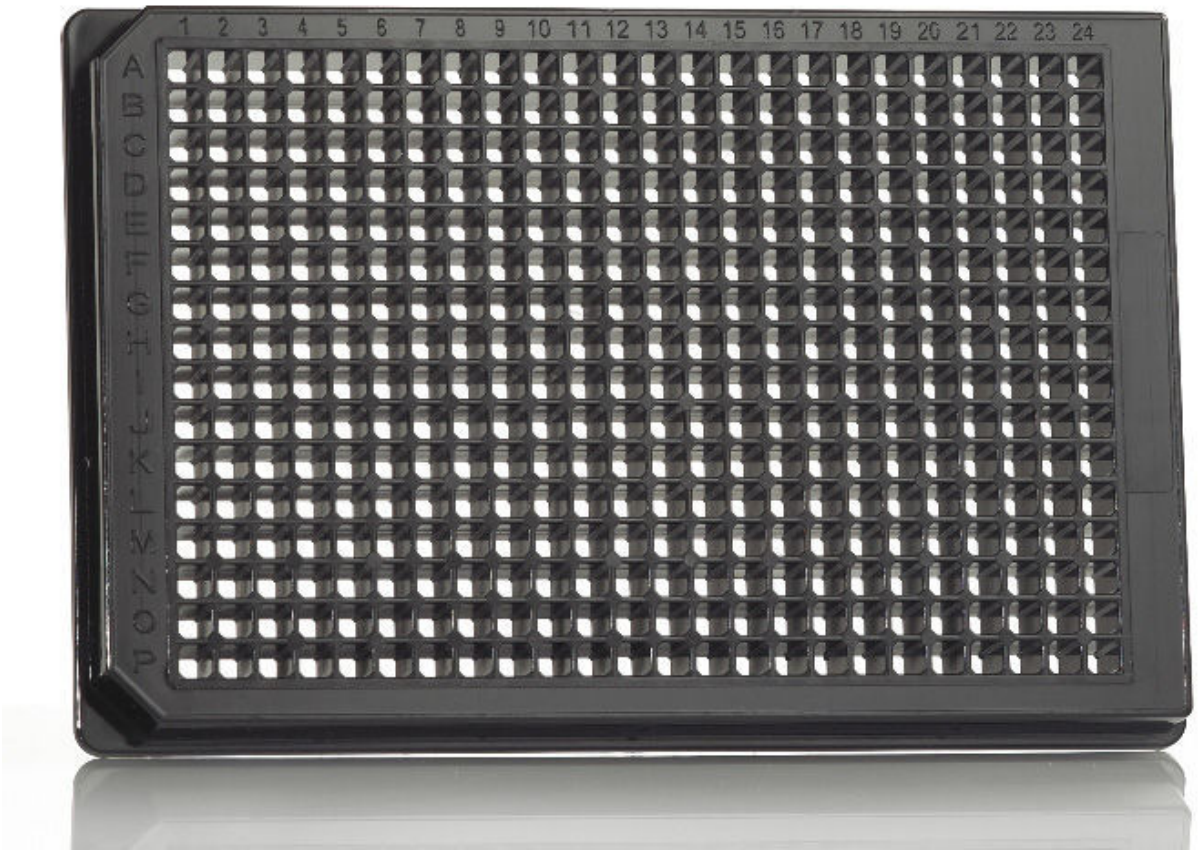


Figure 4: HCS microplate

Each of these microplates are divided into wells (comparable to geographic coordinates) where a dose of a certain drug is pipetted on a collection of cells. In other words, each well allows for a small biochemical experiment. Afterwards, several images of different fluorescent channels are taken, these fluorescent dyes are crucial to understanding the effects of the biochemical experiment as they can be oblivious to the naked eye. Each image can also be divided into coordinates, called fields, which in turn will be analyzed by a cell image processing software package, i.e CellProfiler, and transform these images into numeric

data to be analyzed. The extracted numeric data sets contain up to thousands of metrics called features that describe the phenotype of each cell.

A typical HCS dataset contains both numerical and image data in large quantities and can be representative of the type of data researchers and data analysts can run into in their day to day jobs. That data is getting bigger as time advances, and the need to analyze said data in a timely manner is always needed.

1.4 Research problem and goal:

Data is getting bigger as technology and time advances (Usman et al. 2017), even structured data. Especially in HCS where data is generated from multiple micro-well plates. On average, each plate can generate 50 GB of data, but this can go up to 150 GB. The average research center analyzes tens of thousands of these microplates per year (Mcdonald P, et al. 2008, Omta et al. 2020). Additionally, the increasing usage of HCS has led to data sets that can no longer be processed by biologists due to the complexity of both the data and software used (Omta et al. 2020).
Therefore, biologists had to turn to data scientists to analyze their data. Although multidisciplinary teams have their advantages, the introduction of data scientists has caused delays and multiple iterations of going back and forth between the biologist and the data scientist, which are compounded by the complexity of the biological systems used (Omta et al., 2020). Some solutions are already developed to handle and analyze big data. However, these solutions can have problems like scalability, implementation costs, and compatibility. (Davenport et al., 2012).

Time plays a big role when it comes to data analysis. A good example would be when a global pandemic shook the entire population in 2020 (Haleem et al., 2020). Pharma companies rushed to create a vaccine to save lives and restore normal life with the help of HCS. Furthermore, HCS can be a good representative of what large quantities of structured data could look like as HCS data can be huge when it comes to both image and numerical data.

Based on the previous paragraphs the following main research question for this thesis can be formulated: “How can a framework to analyze structured big data using cloud computing be developed?”

Chapter 2: Theoretical Background

2.1: The need for speed

In 2020, a global pandemic shook the entire population and changed the way humans lived for a long time (Haleem et al., 2020). Time was of the essence, as pharma companies raced to develop a vaccine in an effort to save lives and to return to normal life (Haque, A., & Pant, A. B. 2020). One of the many challenges laboratories faced was the time consuming nature of certain processes that make drug and vaccine discovery possible. HCS was commonly used to study the effects of the SARS-CoV-2 virus on cells as well as testing possible drugs to beat it (Francis R. et al., 2021).

However, even with today's advances, high content screening can still be a slow and tedious process. Both academia and industry have different screening infrastructures. However, these screening methods depend on instrumentation and compound allocations for reagents and data reading and output. Usually academic research centers have a stricter budget and limitations to resources compared to the pharmaceutical industry.

Academic and industry screening centers can scan upwards of 40000 compounds per day with an average of 20000 compounds per day in the most common 384 well format (Major J, 1998 & Mcdonald P, et al., 2008). On the other hand, some pharmaceutical companies can scan more than 100000 compounds per day in high density 1536-well formats (Mcdonald P, et al., 2008). Therefore on the lower average, the industry would need to analyze 20000 compounds in 365 days divided into 384 wells, that's roughly 38020 microplates a year. Each well in a microplate is divided into 12 fields and is usually screened over 5 channels. Pictures of each channel are then taken to analyze the effect of a certain compound over the cells with an image size of about 2.2 MB (Omta et al. 2020), resulting in around 1837[1] TB worth of images to analyze per year, excluding numerical data. Each microplate can then produce around 7 GB of numeric data, resulting in 259 TB of data per year. Therefore, on average, each screening center would have to process around 2096 TB of both numeric and image data per year.

[1] 40000 compounds * 365 / 384-well microplates ~ 38020 microplates per year. 38020 * 384 wells * 12 fields * 5 channels * 2.2Mb (average HCS image size) ~ 1837 TB of image data per year.

2.2: Cloud Computing and Hadoop

Big data analytics and cloud computing have been two of the most ground-breaking technologies to enter the mainstream IT industry in recent years. In fact, these two technologies complement each other quite well, resulting in benefits for multiple disciplines and domains like economics, life sciences, applied sciences, and much more.

Big data is proving to be more and more useful, data can be collected from a multitude of devices and analyzed to either help with decision making, develop new strategies, innovation or even fraud detection. However, big data analysis requires a significant amount of computing resources. A solution to the computing resources problem is deploying big data analytics in cloud computing (Balachandran, B. M., & Prasad, S., 2017).

Big data analytics starts with the acquisition, cleaning, and distribution of the data. In the earlier days, Apache Hadoop was the preferred solution to the traditional big data analytics problems. Apache Hadoop is an Open-source software framework for storing and processing large data-sets on clusters of hardware.

Hadoop has two main components: Firstly, the Hadoop distributed file system (HDFS), mainly used to store large files, and secondly MapReduce which lies at the heart of Hadoop.

Mapreduce is in charge of performing two different tasks in Hadoop programs. The first job is to map, in other words, it takes a collection of data and transforms it into another set of data. After transforming the data, Mapreduce then breaks it into tuples, with key/value pairs, shuffles and reduces the data and then outputs a final result.

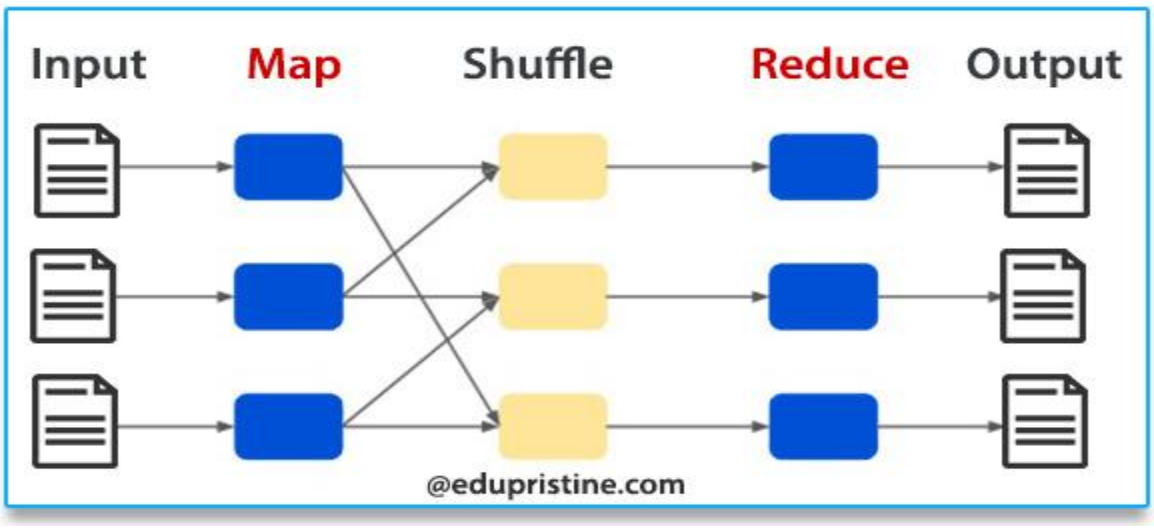


Figure 5: Simplified Mapreduce Workflow Example. Extracted from www.edupristine.com

In their first iterations, Hadoop and MapReduce presented several problems. The main issues were storage and accessibility. However, with the current versions, cloud computing support alongside various data management techniques have been used to reduce the performance gap. Nevertheless, using Hadoop still has some disadvantages to this day, including (Jayasree, M. 2013):

- Restrictive programming model due to prevention of central data
- HDFS is designed in such a way that it doesn't work with random reads on small files because of its optimization for sustained throughput.
- Managing clusters is hard in operations like debugging, distributing software, collection logs etc.
- Because of the single-master model, it requires constant maintenance and may limit scaling.
- Hadoop offers a high security model, but because of its complexity it is hard to implement.
- MapReduce is a batch-based architecture which means it doesn't allow for real-time data access.

2.3: How can HCS images be analyzed?

CellProfiler, a software solution for capturing and analyzing cell images and then transforming the analyzed cell images into numerical data for further statistical analysis. CellProfiler has been the most widely used tool for HCS since 2006. That is because CellProfiler is easy to use, and can be freely downloaded from the CellProfiler website for Windows, Mac, and Unix. It is capable of handling hundreds of thousands of images and contains algorithms for image analysis. The algorithm can accurately identify crowded cells and non-mammalian cell types. CellProfiler was designed and optimized for the most common 2-D high content screening image format. However, researchers interested in time lapse and three-dimensional image analysis were able to build modules to do so because CellProfiler is an open source software (Carpenter, A.E. et al, 2006).

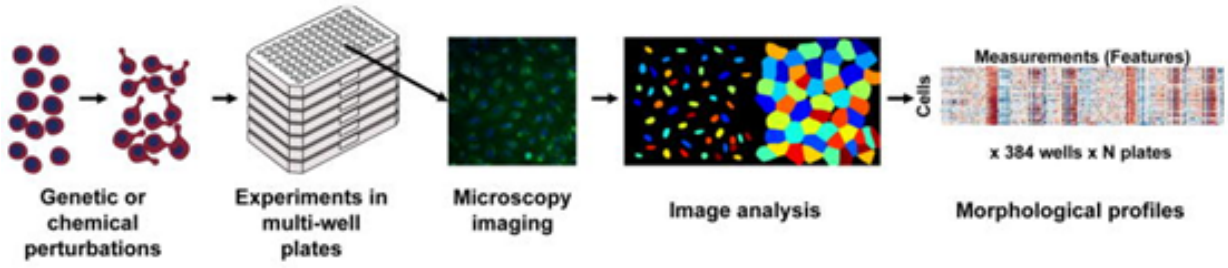


Figure 6: Simplified CellProfiler Workflow (Carpenter, A.E, et al,2006)

Although CellProfiler is the go-to when it comes to HCS, it presented some drawbacks when it came to scalability and making use of multiple cores. Additionally, HCS is a slow procedure in nature due to the sheer size of the data, and although using CellProfiler made it faster, it was still noticeably slow. However, it was still widely used simply for the fact that CellProfiler excelled at bridging the gap between advanced image analysis algorithms and scientists who lack computational expertise (Chakroun, I et al., 2018).

Once the images have been analyzed, CellProfiler creates numeric data from the images. The numeric data can be fed into softwares like StratoMineR for further statistical analysis (Omta et al,. 2016).

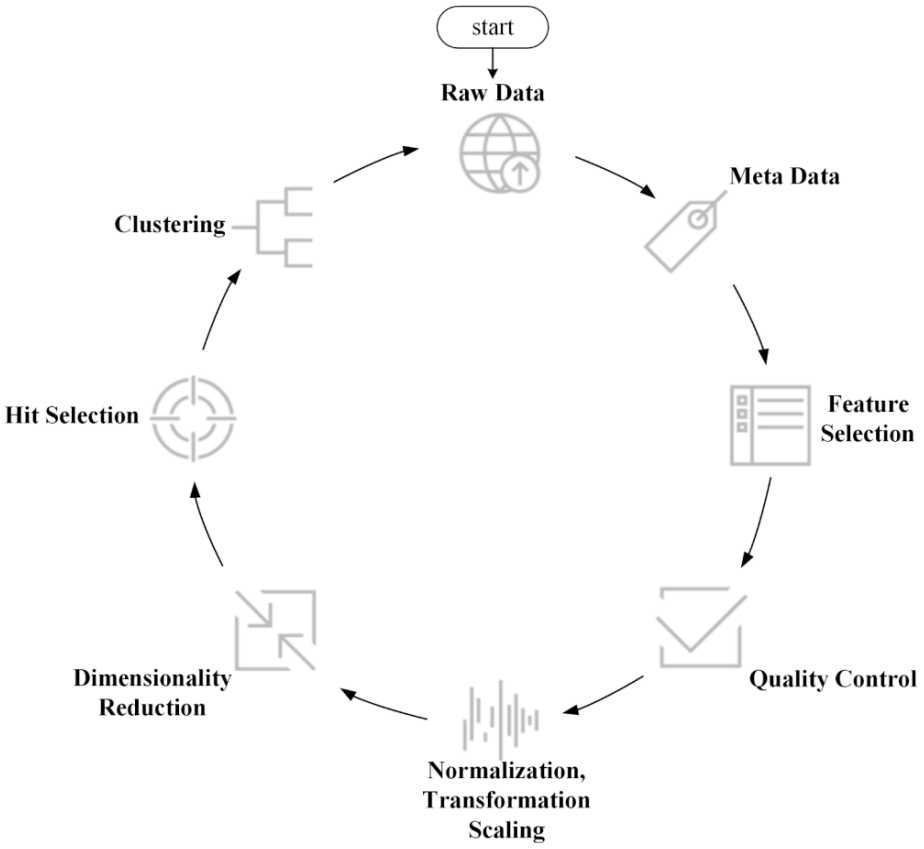


Figure 7: StratoMineR Workflow

Figure 5. explains the workflow of StratoMineR, a web-based and cloud-deployed commercial software package for the analysis of HCS data. The workflow starts with the raw data that can be uploaded by a biologist. The Meta Data section allows the user to configure the system in such a way that any dataset from any vendor can be prepared for the ETL (Extract, Transform, Load) step which is part of the Feature Selection step. Here, the data is fully prepared in the right structure and data type. Certain features are removed that are redundant or contain high amounts of missing data. Quality Control allows

the biologist to perform simple checks in order to determine if the quality of data is acceptable. Then Normalization, Transformation, and Scaling are required steps to prepare the data for downstream analysis steps. Dimensionality Reduction allows for avoiding high dimensionality and redundancy. Dimensionality reduction avoids a biased analysis and reduces the required computational time for multivariate data analysis processes. Hit Selection allows for selecting reagents that show a significant phenotype. Then, Clustering allows reorganizing, grouping, and re-ordering of the data that demonstrates a phenotype into consecutive groups with similar profiles(Omta et al., 2016).

Although the workflow consists of eight steps, the feature selection and hit selection steps usually make up for around 50% of the entire process time (Omta et al., 2016), as they deal with most of the data bulk and include heavy computationally intensive tasks. On that note, it is safe to say that if the time it takes for these steps is minimized, then the overall workflow will also be tremendously affected.

StratoMineR was chosen as reference since this thesis is part of a graduation project at Core Life Analytics, a company that offers cloud computing and data analysis consulting specialized in the life science domain. StratoMineR is Core Life Analytic's main software solution to numeric HCS data analysis.

2.4: Apache Spark

High content screening is naturally an embarrassingly parallelizable problem. HCS is simply docking and scoring libraries of ligands (molecules coordinated to a central atom or molecule in complex) against target proteins by analyzing images of the whole process. HCS analysis and calculations are usually carried out on powerful computer clusters or on large workstations in a brute force manner, mainly by docking and scoring all available ligands. In 2018 Ahmed, L. et al. conducted a study using conformal prediction based virtual screening (CPVS), a machine learning method, in Apache Spark to try and reduce the number of ligands docked and still get an accurate analysis (Ahmed, L. et al., 2018).

The main reason Spark was used over MapReduce is because of its agility as it keeps the data in-memory with support for iterative processing. In an even earlier study in 2017, Ahmed, L. et al. also showed that Google's MapReduce has some limitations. MR is based on a data flow model that penalizes many popular applications where the same dataset needs to be accessed in multiple iterations, i.e machine learning and graph algorithms. In fact, the lack of features like dataset caching, accumulators, broadcast

variables, and native workflows support, makes it hard to develop scientific applications using Hadoop MapReduce (Ahmed, L. et al., 2017).

Apache Spark is an open source cluster-computing framework that overcomes the limitations of MR, while retaining scalability and fault tolerance. Another advantage of Spark is the scalable machine learning library that includes a wide array of regression, classification, clustering and collaborative filtering algorithms. A selection of tools such as featurization, machine learning pipelines, statistics and linear algebra utilities are also available. Although Ahmed, L et al.'s research was promising, they had many limitations. First, running the Apache cluster presented many limitations in terms of resources and costs. They were unable to fire up more than 80 nodes to run their ML model and were only able to achieve a 3.7 times speedup compared to the traditional computational cluster or workstation. However, Ahmed, L et al. 2018 were able to show how processing large datasets in HCS can be time consuming and costly in terms of large compute infrastructures to complete jobs within reasonable time.

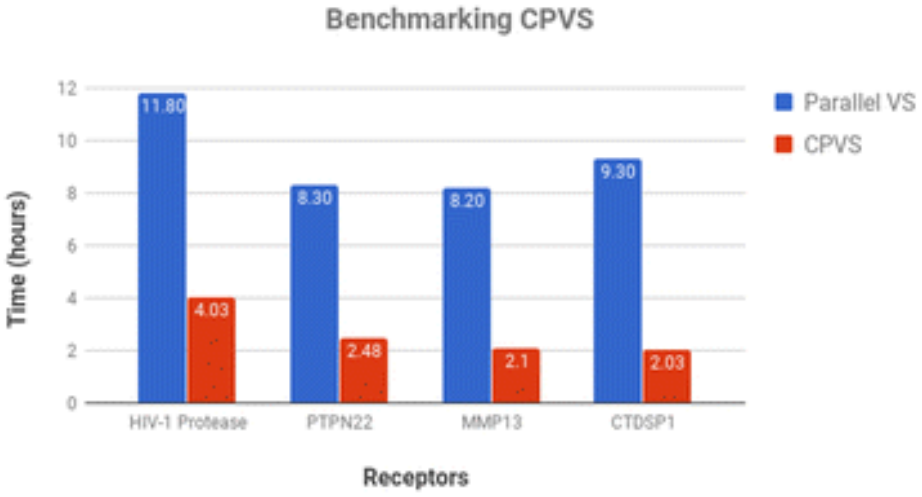


Figure 8: Ahmed, L et al. 2018 paper's results.

The paper concluded with the following statement: "This limited our opportunity for parameter sweeps in the study and necessitated a more tailored approach". In conclusion, on paper, using machine learning within Apache is a possible solution. However, it presents a lot of limitations in terms of scalability and cost.

2.5: The Power of GPU processing

CellProfiler lacked the high performance capabilities needed for HCS where workloads reach hundreds of TB of data and demand computationally powerful workstations. To solve this problem, an experiment was conducted in 2018 by Imen Chakroun, Nick Michiels and Roel Wuyts where they

introduce a GPU-accelerated CellProfiler executing some of the most time-consuming algorithmic steps on Graphics Processing Units. According to the authors of this research paper, CellProfiler was mainly used because of its open-source nature, wide array of biological analysis and ever-growing user base. CellProfiler has been cited in thousands of papers and has won the 2009 Bio-IT World Best Practices Award in IT & Informatics.

As previously mentioned, companies nowadays are capable of scanning and testing upwards of 100000 compounds per day. And as technology advances, image resolution resulting from HCS images are also getting better and bigger in size, therefore identifying objects in those images and analyzing them is becoming more and more computationally intensive and time consuming. In theory, analyzing images from independent samples is an embarrassingly parallel problem that can be sped up by allocating more compute nodes to the analysis. However, up until recently, this was very costly. To put things into perspective, in 2018, a CellProfiler pipeline that extracts 1400 features per cell applied to 7.5 million images averaging in around 30 TB of data was tested. That process lasted about 14 days on 16 compute nodes with 24 cores (Chakroun, I et al., 2018). For Chakroun and her colleagues' research, they had to analyze 50 TBs of HCS data in less than 24 hours. GPUs were a good candidate as they are historically known for performing well on image processing workflows, and seemed to be a good cost effective solution.

The research concluded with a GPU accelerated version of two of CellProfiler's modules, MeasureObjectSizeShape and MeasureTexture. With a data set of 50TB, the MeasureObjectSizeShape and MeasureTexture modules would typically take 2.11 days and 7.72 days respectively. However, with the GPU accelerated version, and the same dataset, the MeasureTexture module took 22.56 hours to complete while the MeasureObjectSizeShape took 9.08 hours.

Although the results were very impressive, the GPU accelerated version would only work on modules that handle image processing, these modules consist of only 2 out of the 90+ available modules. Additionally, it was conducted using 1 single GPU and not a cluster of GPUs which limits the research in terms of scalability, as running a cluster of GPUs to analyze data in parallel adds a whole other layer of complexity.

2.6 Literature Gap and Contribution

Big data comes in different sizes, shapes, and forms. Over the years, researchers have been trying to figure out the fastest methods and techniques to analyze it. However, these techniques have their limitations, either they prove to be very costly, time consuming or just not generalizable over multiple domains and usages. For instance, using Spark or Hadoop for High Content Screening still proves to be challenging in terms of cost and scalability. Additionally, using Graphics Processing Units might be the fastest and most cost effective strategy. However, it requires advanced skill in graphics programming (Zuntich. P , 2018). Furthermore, using GPUs for data analytics is still not heavily supported nor reliable while lacking availability and compatibility to different modules and methods.

Based on the theoretical background, structured big data can be analyzed in many different methods. These methods solve some of the issues that come with big data analysis, like being too costly to implement, too slow, or scalability. Furthermore, compatibility can prove to be a major issue when it comes to big data analysis as some methods work with image data for instance, but are not compatible with numerical data and vice versa. However, these methods do not solve the four issues mentioned earlier at once, each method solves one or two issues at most but struggles with the rest. This thesis aims to fill the gap or find a way to treat all the four aforementioned issues at once.

Chapter 3: Methods & Materials

The approach taken in this research is based on the Research Design Framework developed by John Latham, 2016. A framework designed to help align the main components of a study to deliver the insights needed to draw credible conclusions. The research design framework is meant to help researchers of all types design a custom research methodology for a particular project. The framework consists of nine components linked to each other. These components are linked to the conceptual framework and are organized into two groups:

- The foundation of the problem, which includes the problems, purpose, research questions, and conceptual framework denoted as the "T" in the figure below.

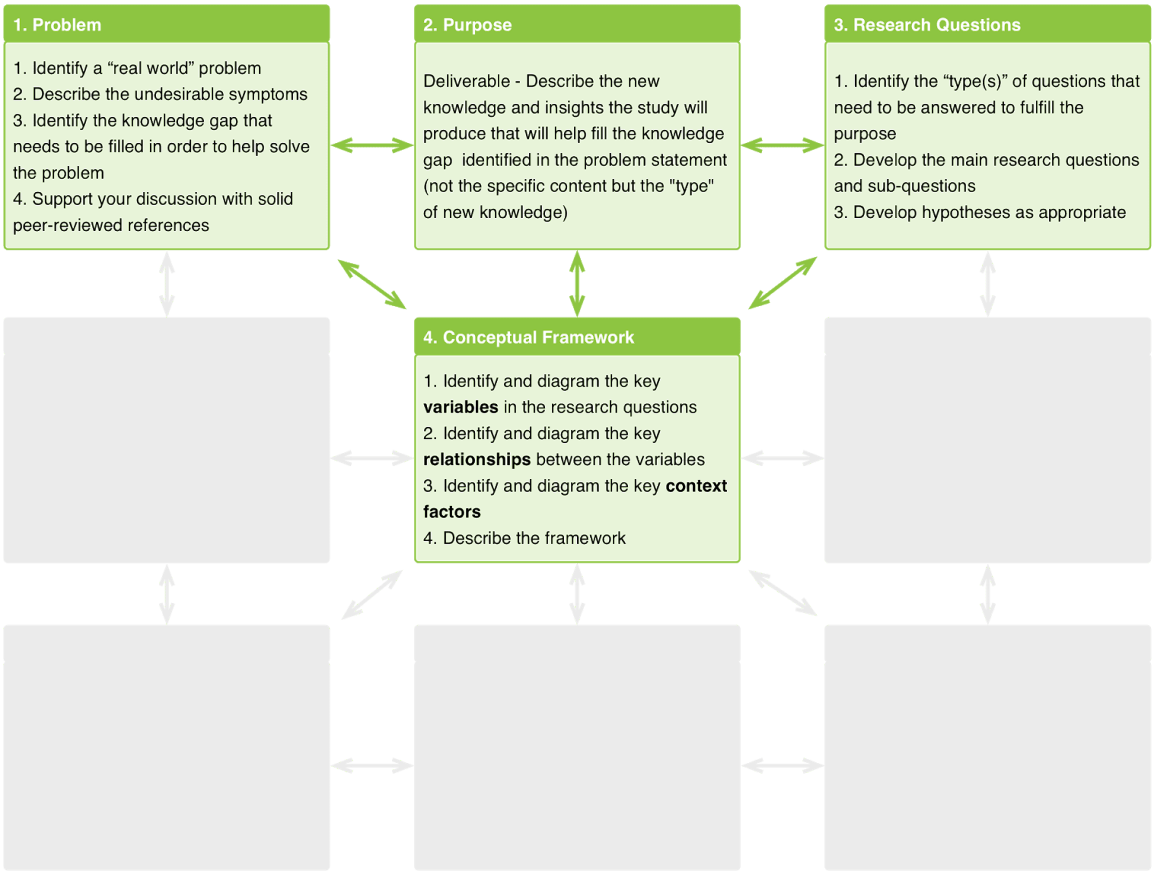


Figure 9: "T" in John Latham's Research Design Framework

- The methodology, which includes the literature review, overall approach data collection, data analysis, and drawing conclusions denoted as the "U" in the figure below.

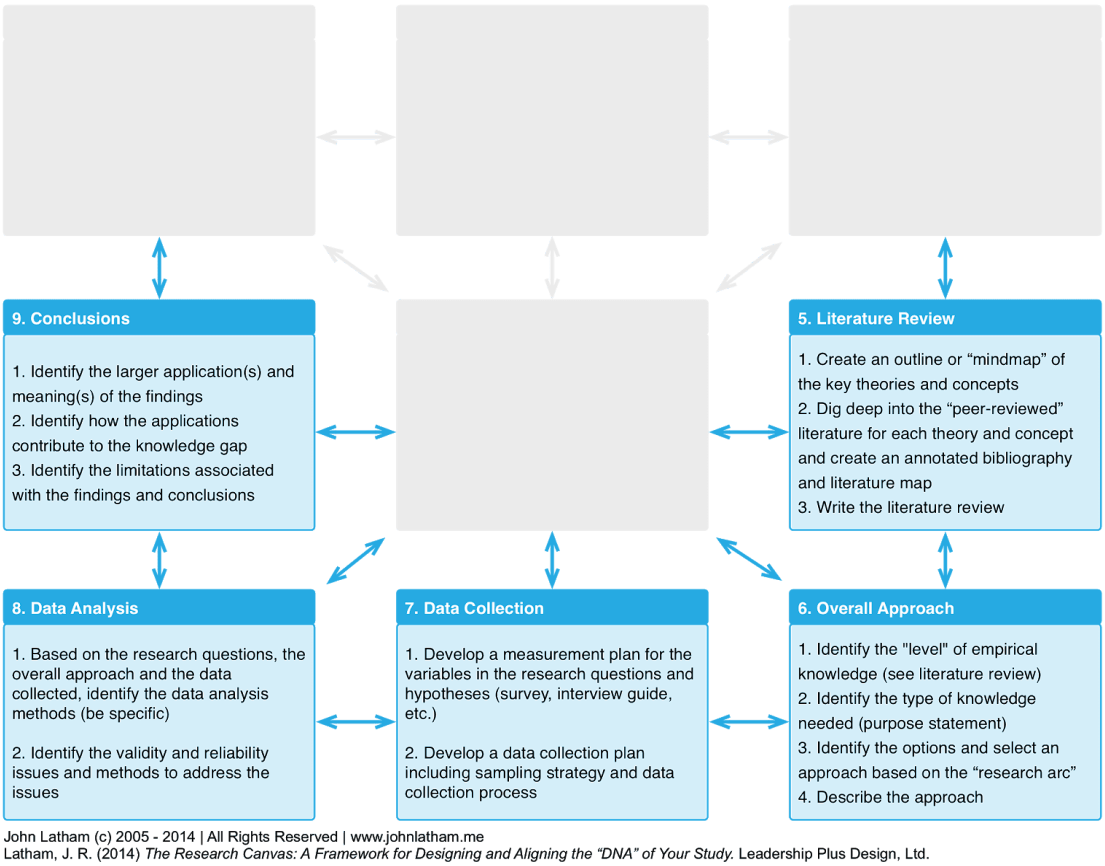


Figure 10: "U" in John Latham's Research Design Framework

3.1 Problem Statement:

It is often the case that the first step in research design is identifying a real-world problem or dilemma. In this particular case, as stated in the introduction, because of big data's variety, it is becoming increasingly hard to analyze said data in a short period of time while keeping costs controlled. Although there are a few solutions that tackle the cost and time problem, new problems like scalability and compatibility arose with these solutions.

After substantial research, it seems that there are no methods or frameworks to analyze structured big data that tackle cost efficiency, time efficiency, compatibility and scalability simultaneously.

3.2 Purpose Statement:

The purpose statement builds on the knowledge gap in the problem statement.

This paper provides a framework to investigate the potential speedup of intensive computational processes within the HCS domain while keeping the costs controlled (cheaper or the same), and maintaining scalability. Furthermore, multiple applications will be deployed using this prototype to test different scenarios so that it can be applied to different domains to ensure compatibility. Ultimately, this thesis aims to build a framework that can generalize the process over other domains and applications.

3.3 Research Questions:

In addition to the main research question "How can a framework to analyze structured big data using cloud computing be developed? ", supporting questions were created to help guide and set the scope of the framework:

SQ1: How can cloud computing make structured big data analysis cost efficient?

SQ2: How can cloud computing make structured big data analysis more scalable?

SQ3: How can cloud computing make structured big data analysis flexible?

SQ4: How can cloud computing speed up the analysis of structured big data?

3.4 Conceptual Framework:

It is common for data analytics and HCS, to own or arrange outsourced computing power in order to perform data processing and analyses in an efficient manner. However, even with the usage of powerful machines, it could take up to months to analyze the data due to its sheer capacity. Therefore, in this paper, we develop a prototype that takes advantage of the scalability of cloud computing to help speed up the HCS analysis process. StratoMineR can be used to research and find a solution to the problem statement. In this thesis, StratoMineR will be adapted to answer the research questions.

Based on the research questions and theoretical background, there are four main components that need to be researched and tested to be able to create the framework. These components are time efficiency, cost efficiency, scalability and compatibility.

Time efficiency and cost efficiency will be researched and tested by mimicking a single-node computing cluster and comparing its costs and performance to a multi-node parallel computing cluster. The single-node and multi-node experiments will be tested by analyzing both image and numeric data in both CellProfiler and StratoMineR using computing instances in AWS.

The performance of these setups will be scored based on the requirements found in the literature study, as well as the following KPIs:

- Cost: the cost of renting the AWS instances
- Completion time: the total time needed to complete the set of tasks

When the benchmarks of both configurations are completed, the scores can be compared using a one-way ANOVA to control type 1 errors since the same data set will be used.

Additional research will be conducted to find a solution and tools for both scalability and compatibility. The solution needs to preferably be within AWS to maintain conveniency and practicality and to keep everything consolidated for ease of use. Once the research questions are answered a framework to analyze structured big data in the cloud can be created based on the research and experiments' results.

3.5 Theoretical Background:

A preliminary research was required to develop the foundation and introduce key concepts required to better understand the thesis. However, to fully grasp the need of this thesis, a theoretical background is needed to better understand and identify the variables, constructs, and relationships identified in the framework and research questions. The preliminary research was based on the literature found via Google Scholar as well as the AWS website.

3.5.1 Create Search Queries:

Search queries were created to find literature related to the paper. These search queries are based on the preliminary research findings. Examples of search queries can be seen in table 1.

SQ1	SQ2	SQ3	SQ4
How can cloud computing make structured big data analysis cost efficient?	*How can cloud computing make structured big data analysis more scalable?*	*How can cloud computing make structured big data analysis flexible?*	*How can cloud computing speed up the analysis of structured big data?*
Cost-effective	AWS instances	Software	Parallelization

cloud computing		Compatibility	
Spot vs On-demand	HCS analysis in cloud computing	Coding in CUDA	Code Optimization
Cloud vs On-site	Amazon S3	Machine Learning versus parallelization	Choosing correct hardware for data analysis
			Data Granularity
			R packages for code optimization

Table 1: Example of search queries for each sub-question

3.5.2 Applying Inclusion and Exclusion Criteria:

Although search queries were created to find relevant literature, the resulting list of potential literature is still substantial. Thus, inclusion and exclusion criteria are needed. These criteria are based on the preliminary research, and describe the conditions that decide whether a paper should be included or not. The criteria are as follows:

Inclusion Criteria

- Papers and studies that are written in English
- Peer-reviewed papers
- Journal Papers
- Not more than 10 years old (with minor exceptions)

Exclusion Criteria

- Patents
- Studies not related to the research questions

3.5.3 Analyze Articles:

The resulting list of literature, after creating the search queries and applying the criteria, is analyzed using NVivo, a qualitative data analysis software used to analyze articles using a technique

called “Coding” (Basley & Jackson, 2013). Coding an article will result in all relevant materials being labeled and indexed.

3.6 Overall Approach:

Based on the purpose of this study and the current knowledge of the research questions, a multi-case computational experiment with a quantitative approach will be executed. According to Yin RK. 1984, a multi-case study requires replication logic rather than sampling logic. In other words, upon finding a significant result from a certain experiment, the immediate goal after that is to replicate the same experiment and re-evaluate the results. Some subsequent experiments might have the same conditions as the initial experiment, whereas other replications might alter a few experimental conditions to check if the finding could still be duplicated. Thus, only with such replications would the original finding be considered robust and worthy of continued investigation or interpretation.

With the previous paragraphs in mind, this thesis’ experiment will be conducted on a control group (the single-node set up), and an experimental group (multi-node set up). These setups were chosen as the single node setup represents analyzing data on a single machine in series, while the multi-node setup represents analyzing data in parallel in the cloud on multiple machines. While parallelization can be done on a single CPU using multithreading it lacks scalability as it is limited to the number of cores the CPU has. Therefore, the power of parallelization can be utilized to its fullest when it is paired up with the theoretical infinite scalability of cloud computing. The experiment will also have varying multiple experimental conditions (the mixture of core count and machines). The experiment will then be re-run multiple times and then an average of the KPIs (Time and Costs) will be measured and evaluated to draw conclusions.

3.7 Data Collection:

According to Latham. 2014, data collection planning usually consists of three key components: a sampling plan, a measurement plan, and a data collection plan. In this paper, the sampling plan consists of the two experimental set-ups (single-node versus multi-node). Furthermore, the measurement plan consists of using StratoMineR, and programming languages like R, specifically the system.time() function alongside packages like the microbenchmark library to measure reproducible completion times. Additionally, the AWS web app will be used to manage nodes and keep track of costs, CPU usage, and other useful metrics that will be discussed further in the paper. As for the data collection plan, some scripts and sample data will be provided by Core Life Analytics as this thesis is a graduation project in collaboration with them.

3.8 Data Analysis:

Measurement and data collection are usually focused on constructs, factors, variables, and context. Data analysis, on the other hand, is generally focused on the relationships between these objects. In this paper, the relationship between the constructs and variables will be analyzed to answer the research questions.

3.9 Drawing Conclusions:

The last step in a research process is putting all the pieces together in a coherent discussion of key findings and their implications for theory and practice. Therefore, after the data analysis step, a discussion chapter will be introduced to draw a conclusion based on the results. Additionally, further research options will be discussed.

Chapter 4: Framework Research, Experiment Setup, and Results

4.1 Framework Introduction

As described in the problem statement, there are currently no mainstream methods or frameworks that solve all the issues associated with structured big data analysis at once. These issues are mainly scalability, compatibility, cost efficiency, and time efficiency. For example, CellProfiler is a software solution to analyze HCS image data, but CellProfiler does not work with other forms of structured data. Additionally, GPU acceleration can be used to analyze data extremely fast. However, it only works on image data and is not widely supported in many applications as it is not commonly used. Additionally, GPU acceleration requires advanced coding skills. Furthermore, Solutions like Spark, MapReduce and Hadoop can be viable in certain situations but can present drawbacks in scalability and cost effectiveness.

Some research was needed in order to develop a framework to possibly solve the problems mentioned above. Additionally, two experiments were conducted to test performance and cost efficiency. Those experiments were run on Amazon Web Services since AWS had some tools that made the experiment easier to fulfill and maintain scalability and compatibility when needed. Additionally, since AWS has the largest market share, it offers more availability and a larger selection of instances than most of its competitors. However, any cloud computing provider that can provide similar tools and instances can be used. Furthermore, AWS was used as it has the bigger market share, meaning that a wider selection of nodes are more readily available. More details will be given in the upcoming sections of this chapter.

4.2 Scalability

To ensure scalability, two factors needed to be taken into consideration, storage and computing power. When it comes to computing power, as mentioned in the introduction, AWS offers a wide array of Amazon Elastic Compute Cloud (EC2) instances that are highly available. As for storage AWS offers

their Simple Storage Service and Elastic Block store that are very scalable. More details will be revealed in the upcoming sub-sections

4.2.1 Amazon Elastic Compute Cloud

The Amazon Elastic Compute Cloud (EC2) is on-demand computing power that allows subscribers to rent virtual machines in the cloud with no long term commitment. It enables users to boot up new AWS virtual servers in minutes and rapidly scale up or down. AWS supports Windows, Linux, macOS, FreeBSD, and Open Solaris. Additionally, Amazon EC2 can be used with all major Web and application platforms. An Amazon EC2 environment includes the operating system, services, database, and application platform stack required for a cloud-hosted application service. The virtual application stack can be started, stopped, restarted, or rebooted from a Web-based console using Web service APIs, with 99.95% availability per region using Availability Zones (Whitehouse, L., & Buffington, J., 2012). As a testament to AWS' scalability, the Financial Industry Regulatory Authority (FINRA) in the USA has issued a statement in 2021 stating that with the help of AWS they were able to automatically boot up and shut down 100000 instances in a single day.

4.2.1 Amazon Simple Storage Service

Amazon Simple Storage Service (S3) is a cloud-based object store available through Web services interfaces such as Representational State Transfer (REST) and Simple Object Access Protocol (SOAP). It is used as a cloud storage container for backup data and images. Users can write, read, and delete virtually an unlimited number of objects ranging from one byte to 5 TB of data each. Amazon S3 is similar to a traditional on-premise Storage Area Network (SAN) or Network Attached Storage (NAS) device. However, as an AWS cloud implementation, it is far more agile, flexible, and virtually available anywhere in the world (Whitehouse, L., & Buffington, J., 2012).

4.2.2 Amazon Elastic Block Store

Amazon Elastic Block Store is Amazon's block-storage service, it is designed to be used with Amazon Elastic Compute Cloud (EC2) for both throughput and transaction intensive workloads at varying scale. EBS supports a broad range of workloads, such as relational and non-relational databases,

enterprise applications, containerized applications, big data analytics engines, file systems, and media workflows are widely deployed on Amazon EBS (https://aws.amazon.com/, n.d).

4.3 Compatibility

As discussed in the previous chapters, compatibility can prove to be a tough challenge, certain data sets require specialist software to be analyzed, others have hardware or operating system requirements. When parallelizing in the cloud, especially on multiple machines, the user might need to set up these machines in a certain way before they can be operational. This can be a tedious task especially if the user needs to set up hundreds or thousands of instances.

In AWS it is possible to back up data on an Amazon Elastic Block Store (EBS) volume to Amazon Simple Storage Service (S3) by taking point-in-time snapshots. Snapshots are incremental backups, which means that only the blocks on the device that have changed after the most recent snapshot are saved. This minimizes the time required to create the snapshot and saves on storage costs by not duplicating data. Each snapshot contains all of the information that is needed to restore the data from the moment when the snapshot was taken to a new EBS volume.

When an EBS volume is created based on a snapshot, the new volume begins as an exact replica of the original volume that was used to create the snapshot. The replicated volume loads data in the background so that the user can begin using it immediately. If the user tries to access data that hasn't been loaded yet, the volume immediately downloads the requested data from Amazon S3, and then continues loading the rest of the volume's data in the background. Amazon Machine Images (AMI) were used to ensure compatibility with different softwares and set ups using the EBS snapshots. An Amazon Machine Image basically provides the information required to launch an instance. The user first sets up a machine, its operating system, required softwares, applications, etc. and then takes an EBS snapshot of their setup and registers it, this is called an AMI. This AMI can be used to fire up instances with the same settings and setup as the machine initially configured. Multiple instances can be launched from a single AMI, this is useful when multiple instances with the same configuration are needed. Different AMIs can be used to launch instances with different configurations as well (https://aws.amazon.com/, n.d).

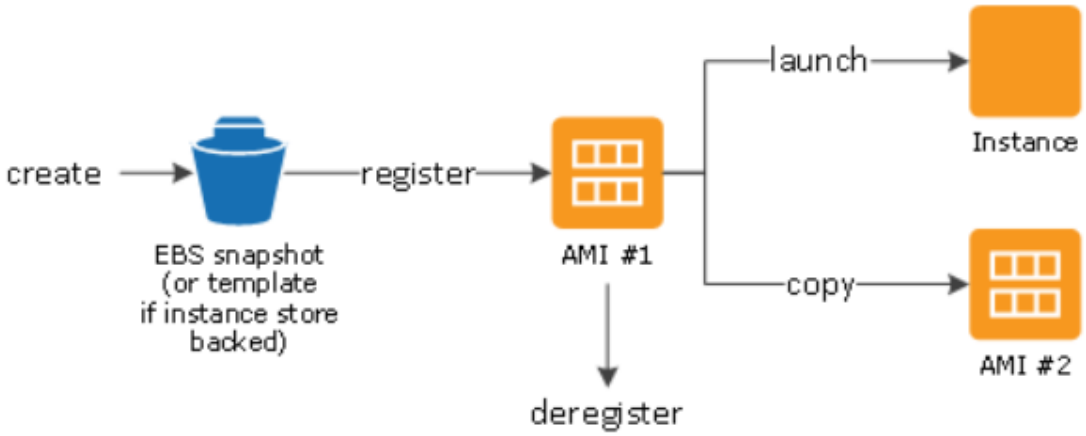


Figure 11: Simplified AMI workflow. Extracted from (www.amazon.com)

For example in this experiment, an AMI was configured to boot instances with a version of CellProfiler to analyze image HCS data while another AMI was configured to boot instances with a version of StratoMineR and other tools like R to analyze numeric HCS data. As soon as the instances would fire up they would automatically install the appropriate applications and software in the AMI.

That is why AMIs and the wide selection of AWS instances that vary in operating systems and hardware can prove to be a powerful tool to ensure compatibility with almost any software or data set.

4.4 Time Efficiency

The experiment begins with both a numeric and image HCS data set, HCS data sets were chosen as they can be a good representative of what a typical structured data set could look since it contains both numerical and image data. The purpose of this part of the experiment is to check whether analyzing the data on compute instances in the cloud is faster than doing it locally on a single machine. Multiple machines with different setups will be tested versus similar multi-machine setups in the cloud.

4.4.1 Analyzing Numeric Data Set in StratoMineR

The numeric data set contains cell-level data from 24 micro-well plates in duplicates resulting in 48 plates, each plate has 384 wells and has approximately 444.4 thousand records per plate totalling in 21.3 million records in the entire data set. This data set was previously uploaded to S3 for the purpose of this experiment, but first needs to be queried on to the master node. The master node is also an EC2 instance responsible for getting the data from S3 to either the multi-node setup or the single-node setup. In this case the master node is the node that is running the StratoMineR user interface on the cloud. Once the data is on the master node, it can be uploaded to either one of the setup options in table 2. These instances were particularly picked as they are the most efficient memory optimized Advanced RISC Machines (ARM) instances according to Amazon. Additionally the setups were created this way to try and create the fairest one to one comparison between single-node setups and multi-node setups.

1 machine 4 cores (local)	1 machine 8 cores (local)	1 machine 16 cores (local)	4 machines 1 core (parallel)	8 machines 1 core (parallel)	16 machines 1 core (parallel)
r6g.xlarge	r6g.2xlarge	r6g.4xlarge	4 c6g.medium	8 c6g.medium	16 c6g.medium
4 cores	8 cores	16 cores	4 cores in total	8 cores in total	16 cores in total
32 gbs RAM	16 GBs RAM	32 GBs RAM	8 GBs RAM	16 GBs RAM	32 GBs RAM
4.75 Gbit/s throughput	4.75 Gbit/s throughput	4.75 Gbit/s throughput	40 Gbit/s throughput	80 Gbit/s throughput	160 Gbit/s throughput
$0.12 per Hour	$0.1587 per Hour	$0.3174 per Hour	$0.072 per Hour	$0.144 per Hour	$0.288 per Hour

Table 2: Numeric data experiment node setups

Once uploaded to the required setup, the data undergoes the StratoMineR workflow, but for the purpose of this study only the ETL, Feature Elimination (FE), and Feature Selection (FS) Processes will be measured since as stated previously, the ETL and Feature Selection processes make up for the majority of the StratoMineR workflow in terms of time. Therefore, if there is a significant decrease in completion time in these steps then the overall workflow will benefit greatly. The Feature Elimination Process will also be measured as it is a prerequisite step going from the ETL to the Feature Selection Process.

Each of these setups will analyze the same data set using StratoMineR for 5 times. Additionally, after each run, the instance(s) used were shut down and re-booted to ensure that no data was cached on an instance. Following the data analysis runs, the mean and standard error will be calculated for each of the

StratoMineR steps as well as the overall process. After calculating the mean and standard error a one-way ANOVA test will be conducted to determine the p values for the aforementioned steps.

The results were as follows:

	Local 4 cores (N = 5)	Local 8 Cores (N = 5)	Local 16 Cores (N = 5)
ETL	~247 seconds	~290 seconds	~91 seconds
FE	~0.3 seconds	~0.16 seconds	~0.14 seconds
FS	~508 seconds	~77 seconds	~45 seconds
Overall	~756 seconds	~366 seconds	~146 seconds
Money Spent	$0.0192	$0.0161	$0.012

Table 3: Single-node setups run times for ETL, FE, and FS

	Cluster 4 cores (N = 5)	Cluster 8 cores (N = 5)	Cluster 16 cores (N = 5)
ETL	~217 seconds	~113 seconds	~56 second s
FE	~6 seconds	~5 seconds	~5.6 seconds
FS	~276 seconds	~123 seconds	~58 seconds
Overall	~497 seconds	~241 seconds	~120 seconds
Money Spent	$0.00994	$0.00964	$0.0096

Table 4: Multi-node setups run times for ETL, FE, and FS

	Overall p-value ($\alpha = 0.05$)	ETL p-value ($\alpha = 0.05$)	FE p-value ($\alpha = 0.05$)	FS p-value ($\alpha = 0.05$)
4 core single-node vs multi-node	< .00001	< .00001	< .00001	< .00001
8 core single-node vs multi-node	< .00001	< .00001	< .00001	< .00001
16 core single-node vs multi-node	< .00001	< .00001	< .00001	< .00001

Table 5: Single vs Multi-node setup p-values using one-way ANOVA

The experiment resulted in a mean of around 30% increase in performance to the overall workflow in a multi-node setup. Furthermore, based on the p-values for all the comparisons in table 5, it is clear that a multi-node setup is significantly better than a single-node setup for $\alpha = 0.05$.

4.4.2 Analyzing Image Data Set in CellProfiler

The image dataset used is the "AGM dataset". It is derived from a high content screen published by the Carragher group at the University of Edinburgh in 2010. Four cell-lines were screened against a library of 102 well-annotated drugs and inhibitors. The reagents were run in eight half-log doses, in triplicate assay 96-well plates. The cells were stained with dyes to label the actin cytoskeleton, microtubule and the DNA. The images were acquired on a Molecular Devices ImageXpress 5000A with a 20X magnification. An example of what the images look like is shown in Figure 12.

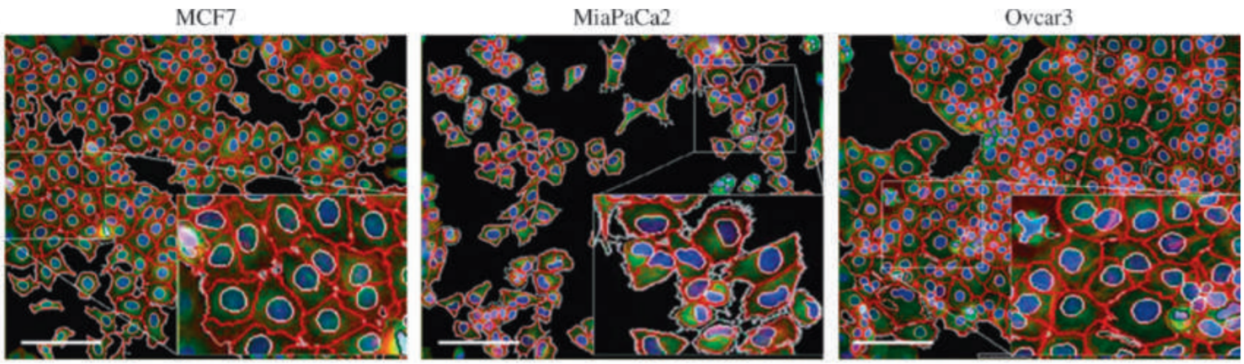


Figure 12: HCS images example from the "AGM" dataset (University of Edinburgh, 2010)

The images for one cell line were loaded into the Image Data Repository, and the Broad Bioimage Benchmark Collection. CellProfiler was used to extract 479 features per cell. The cell-level data generates ~230MB of data per plate. The AGM set in total contains 57 96-well plates of data. In addition, the dataset contains 5 annotated classes (NEGATIVE, Docetaxel, Doxorubicin, LatrunculinB, A Z). This data is from one 96 well plate containing 240 images across 3 channels resulting in 720 image sets. These sets translate into 503 batches that need to be analyzed by CellProfiler.

Similar to the numeric data analysis, the image data set will be analyzed in CellProfiler using multiple setups. The main setups to compare are again single node machines with powerful specifications, versus multiple small machines with single cores. However, since CellProfiler currently does not work on ARM architecture CPUs, different instances and setups were chosen to run the experiment. Additionally, because of some time constraints, the analysis time of only 5 batches was measured on each of the set ups

in table 6. The analysis was repeated 5 times and the mean analysis time was calculated to compare the set ups.

1 machine 1 cores	1 machine 2 cores (laptop)	1 machine 4 cores	1 machines 16 cores	1 machines 36 cores	1 machines 72 cores
t2.small	laptop	c5.xlarge	c5.4xlarge	8 c6g.medium	16 c6g.medium
1 core	2 cores	4 cores	16 cores	36 cores	72 cores
2 gbs RAM	8 GBs RAM	8 GBs RAM	32 GBs RAM	72 GBs RAM	144 GBs RAM
4.75 Gbit/s throughput	300 Mbit/s	4.75 Gbit/s throughput	10 Gbit/s throughput	10 Gbit/s throughput	10 Gbit/s throughput
$0.0250 per Hour	N/A	$0.192 per Hour	$0.768 per Hour	$1.728 per Hour	$3.456 per Hour

Table 6: Image data analysis setups

CellProfiler runs image analysis tasks in series the majority of the time, there is an option to run image analysis in parallel, however the number of images that need to be analyzed have to match the number of cores available in a machine. Additionally, the multithreading option is only supported by a few out of the 90+ modules that CellProfiler has since it's an open source software, and most of its modules are written by the community. Therefore, a decision was made to compare running CellProfiler in parallel on multiple 1 core machines in the cloud versus a local laptop limited to 2 cores (to have a fair comparison) and versus the single, multiple core, machines mentioned in table 7.

Instance	5 batches runtime (N = 5)	Money Spent	Estimated Total runtime for 500 batches	Estimated Total Money Spent
t2.small	~310 seconds	$0.002152778	~310 seconds (parallel)	~$0.215 (for 100 instances spun up)
c5.xlarge	~387 seconds	$0.020640000	~38700 seconds ~ 645 mins (series)	~$2.064
c5.4xlarge	~349 seconds	$0.074453333	~34900 seconds ~ 582 mins (series)	~$7.4496
c5.9xlarge	~357 seconds	$0.171360000	~35700 seconds ~ 595 mins (series)	~$17.136
c5.18xlarge	~351 seconds	$0.336960000	~35100 seconds ~ 585 mins (series)	~$33.696
Local laptop (2 core)	~1380	N/A	~138000 seconds ~ 2300 mins (series)	N/A

Table 7: Image data analysis results for each setup

The experiment shows us that 5 batches take roughly 6 minutes to be analyzed on CellProfiler regardless of core count. The single machines with multiple cores need to run the remaining batches in series, therefore, the entire data will take approximately up to 10 hours to complete and cost anywhere between $1.92 and $34.5. However, the machines with a single core will take roughly 6 minutes and cost around ~$0.215 to analyze the entire data set with 100 instances since the machines are running in parallel in the cloud. Additionally, the increased analysis time on the local laptop can be explained by the thermal throttling of the CPU and poor heat management. Therefore, the laptop CPU could not perform as good as it normally would in optimal conditions.

4.5 Cost Efficiency

As previously mentioned in the introduction, there are two ways to fire up instances in AWS, on demand and spot pricing. A user can pay to get immediate access to a computing instance and pay per hour (On demand), or the user can bid to rent instances from AWS' spare capacity and get a discount from on-demand pricing (Spot) at the risk of having the instance terminated from AWS at a certain moment. For example an a1.medium instance costs $0.0288 per hour on demand whereas it can go as low as

$0.0089 per Hour in spot pricing. The underlying hardware is the same, there's no difference in performance yet prices differ greatly.

In this experiment 80 a1.medium and 1 r6g.16xlarge ran in both spot and on-demand pricing were given a data set to analyze and put under load for 12 minutes. The purpose of this experiment is to determine how cost efficient parallelization in the cloud can be, given the same hypothetical analysis times.

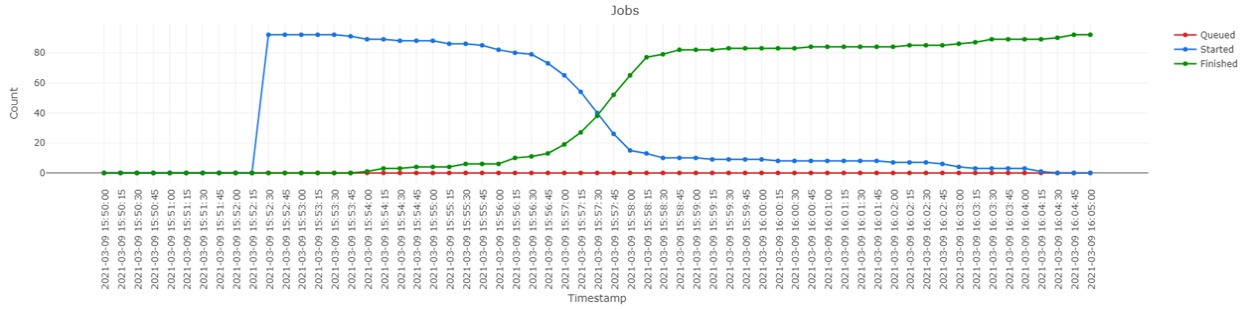


Figure 13: Queued and Completed Jobs in CellProfiler

In figure 13, the blue line represents the analysis jobs or batches the instances need to do. The green line in return represents the amount of finished jobs. At the first glance, it is noticeable that the 80 a1.medium instances cost $0.142464 and $0.4608 in spot and on-demand respectively while the r6g.16xlarge is $0.2544 and $0.72192 in spot and on-demand respectively. It is quite clear that for both cases using spot pricing is the most cost efficient solution. Nonetheless, the more remarkable point is that 80 a1.mediums were 56% cheaper than the r6g.16xlarge in spot pricing under the same hypothetical workload. However, there are a few things to take into consideration in this experiment:

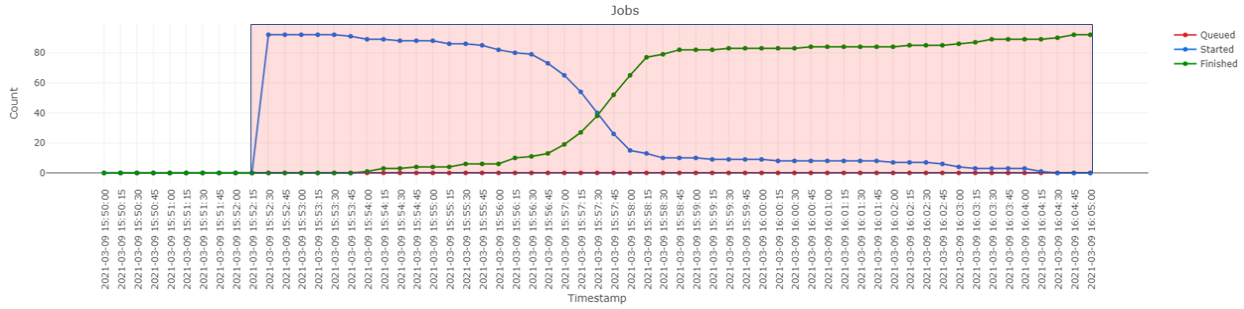


Figure 14: Single-node Setup Costs AOU Representation

Firstly, as shown in figure 14, as the number of jobs starts to decrease the load on the machine is decreasing as well. However, the user is still paying for the full 64 cores for the duration of the analysis

task as depicted by the red area in the graph. This means that roughly 45% into the data analysis process the machine is not being used to its fullest potential.

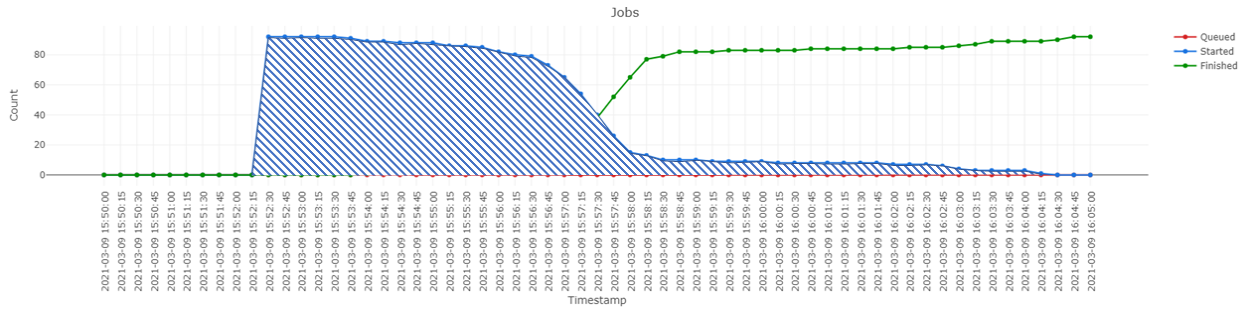


Figure 15: Multi-node Setup Costs AOU representation.

Secondly, as shown in figure 15, in a multi-node setup, as soon as an instance finishes its given task, it can be shut down immediately. Meaning that in a real world scenario, there are no lost costs when using a multi-node setup depicted by the hashed blue area in the graph. On that account, if the area under the blue hashed curve is roughly 45% of the red box then the actual spot costs for 80 a1.mediums would be $0.0641088.

Chapter 5: Discussion, Framework, and Conclusion

5.1 Experiment Overview

The experiments and research conducted in chapter 4 show some interesting results. Figure 16 shows how all the previous small experiments come together to form one coherent workflow to analyze structured big data using AWS.

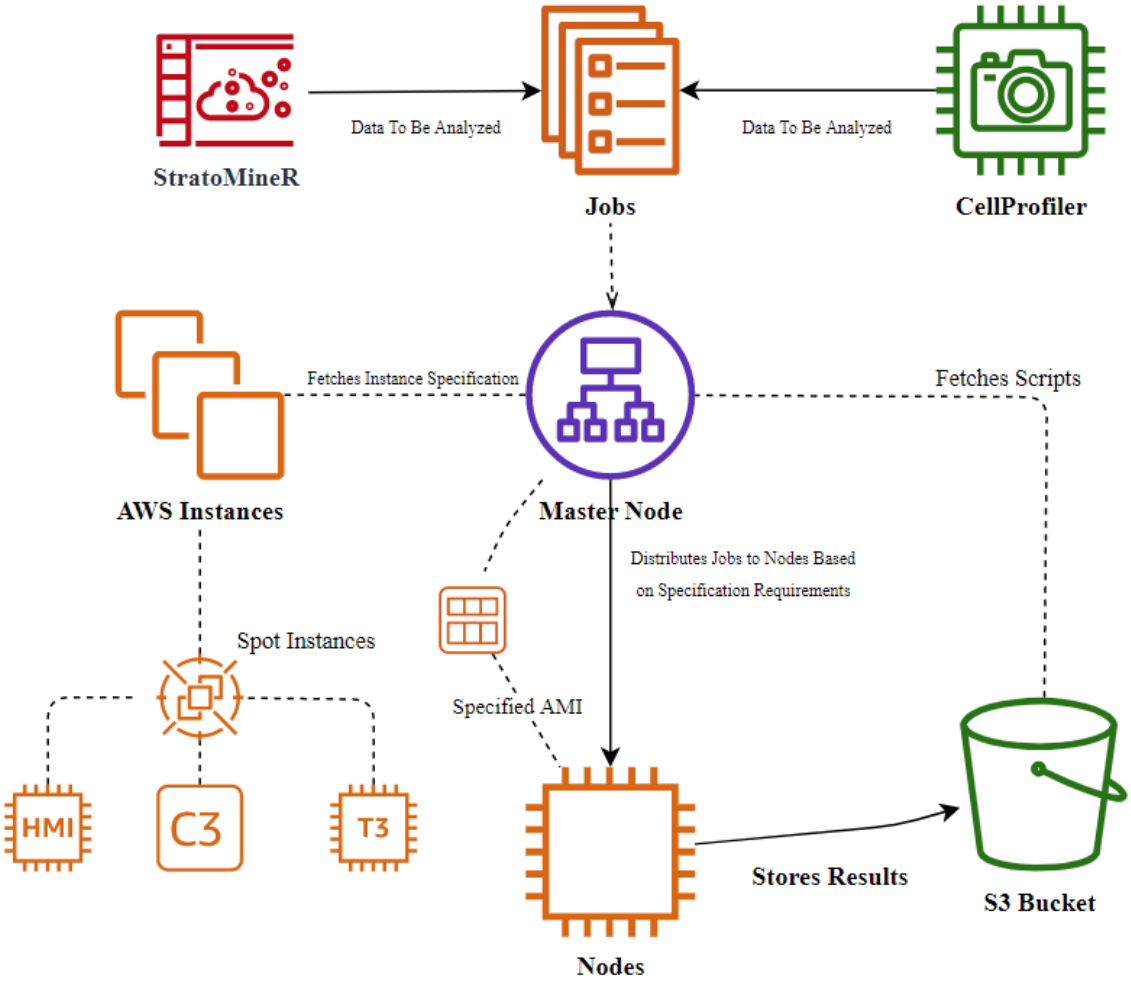


Figure 16: Workflow to analyze structured big data using AWS

The workflow starts with structured big data queried from the user's S3 bucket (Mass storage) containing either numeric data (StratoMineR) or image data (CellProfiler). The data then gets split up into smaller chunks, called jobs, following a certain granularity that the user decides. The data is then sent to the master node where it fetches the necessary coding scripts needed to run the analysis in the cloud. These scripts can contain the instance specifications, the number of instances needed, the split-apply-combine scripts and many more, depending on what the user needs to do. After fetching the necessary scripts, the master node then requests additional instances to be fired up from AWS in spot pricing. These instances are then configured to a registered AMI which also in turn installs the required software needed to run the analyses on the instances in the cloud. Afterwards, the results from the analyses are merged using a merge script with the preferred coding language and stored back in the user's S3 bucket.

5.2 Limitations

The experiments conducted for this thesis presented some interesting results. However, the research and thesis in general still had a few limitations that need to be noted and can prove to be useful for the experiment's reproducibility and future work.

- The most notable limitation is that the experiments conducted in the cloud were embarrassingly parallel problems. Meaning that the data used is structured enough to be divided into batches that are independent of each other. This allows the data to be processed in parallel without having the output of a certain batch analysis as input to another batch of data. This could prove to be a problem to compatibility as it can be extremely hard to analyze unstructured big data in parallel using the same methods in this thesis' experiments.
- Although this thesis' experiments provided ways to deal with compatibility by using AMIs in AWS, it was still only tested using StratoMineR and CellProfiler. Meaning that no other applications or softwares were tested to ensure compatibility with everything.
- All the instances used and tested in AWS were located in Ireland. Therefore, the availability and spot prices of other areas might differ. However, AWS is usually consistent with pricing and availability.
- Instance boot-up and shut down times were not measured, which adds up to the total analysis times in the cloud. However, on-demand instances power up almost instantly while spot instances might take 30 to 60 seconds to boot up. Nonetheless these numbers were not logged and may seem rather arbitrary.

- The data sets used were not very large in terms of raw size. The numeric data set was around 12 GBs while the image data set was around 7.5 GBs. Nevertheless, it's expected to have even better results with a larger data set as analysis times increase greatly. However, in certain cases the analyzed data might surpass the instance's local storage capacity which might cause some problems. A quick fix would be using EBS but it has not been tested in this thesis as a larger data set was not readily available.
- S3 storage and EBS pricing was not taken into consideration. Nonetheless, a quick search revealed that the average price per GB on an enterprise Hadoop or spark cluster is between $1000 to $2000 depending on the desired performance. Whereas the price per TB on S3 averages around $24 excluding EC2 instances.

5.3 Discussion and Framework

Taking the experiment results and limitations into consideration there are a few things worth mentioning and discussing.

5.3.1 Discussion

Firstly, although boot up times were not logged, they are still very important to take into consideration. Boot up times, individual batch completion times, and data size play a fundamental role in deciding if a data set should be analyzed in parallel in the cloud or in series on a single machine. For example, in section 4.4.1 the feature elimination step actually performed better on a single machine than on the cloud. It first has to wait for the ETL process to be done, aggregate the data and then run the feature elimination step. Using multiple machines on the cloud, the feature elimination step was actually 20 times slower on average. Although it was only a matter of split seconds, combined with the instances' boot up times, it might actually make a great difference in choosing whether to analyze in parallel or series since it also takes some time for the nodes to pick up the jobs and start the analysis. Therefore, a possible solution to this problem is to actually do some testing beforehand and measure how long an analysis batch actually takes and compare it to boot up and pick up times and come up with some sort of ratio. If a batch takes longer to analyze than the node boot up and pick up times, then it's probably better to run the analysis or task in the cloud, otherwise it might be better to just do it in series (more details and an example in appendices).

Secondly, the experiment design for the data analysis step was created in a way to make the comparison as direct and fair as possible. However, in practice, the strength of cloud computing lies in the ability to fire up hundreds of nodes to tackle a problem instead of nodes equivalent to the single machine set up. For example in section 4.4.1 only 16 machines were fired up to make the comparison as fair as possible, yet in practice it would have been possible to fire up nodes and match each batch or job to be completed by a node, resulting in 48 nodes fired up. This however means that the analysis time was actually decreased from 146 seconds down to 38 seconds on average while only spending $0.010032 instead of $0.012. Although the user is firing up and using more nodes, the reduction in overall analysis time also reduces or roughly maintains the overall cost.

Thirdly, taking both previous paragraphs into consideration, cost plays a fundamental role in running tasks in parallel in the cloud. In theory, a user can fire up thousands of instances in the cloud to run certain tasks in parallel. However, that is only applicable if the user has an infinite amount of money and no budget constraints. Therefore, for future usage of this paper's method (or framework) a budget needs to be specified beforehand to determine the amount of instances a user can fire up and to make sure that the amount of instances used is actually going to make the process faster by a significant margin.

Finally, taking budget, batch completion times, node start up and pick up times into consideration alongside the previous methods, a framework can be developed and used to analyze structured big data in the cloud.

5.3.2 Framework

These are the steps shown in figure 16 for the framework:

- Step 1: Decide data granularity
    - This step is crucial as it can reduce the size of the data, without losing accuracy, and have a major impact on the analysis computing performance
- Step 2: Upload data into mass storage.
    - Preferably a mass storage service similar to S3 to ensure scalability
- Step 3: Decide batch size, budget, and purpose
    - Based on the batch sizes and budget, it is possible to determine the type and number of instances needed to run the analysis in parallel in the cloud
    - Based on the purpose, there is a vast selection of instances to choose from. For example there are memory optimized, compute optimized and even GPU instances to choose from.

- Step 4: Fire up required instances using AMI or similar technology
    - Using AMIs ensures compatibility as it will pre-install the required tools on the instance before it runs the assigned task. However, AMIs need an initial configuration which might take some time, but it enables the user to fire up thousands of instances automatically and install the required tools with a single click.
- Step 5: Submit Data to computing cluster in the cloud
    - This step might prove to be tricky, but if the user follows Hadley Wickham's Split-Apply-Combine method (Previously discussed in the thesis) then it should be easily possible.
    - This step can be realized and done using the preferred scripting language
- Step 6: Combine Analysis Results and Re-upload aggregated results to mass storage
    - Again, this step can be done using the preferred scripting language.

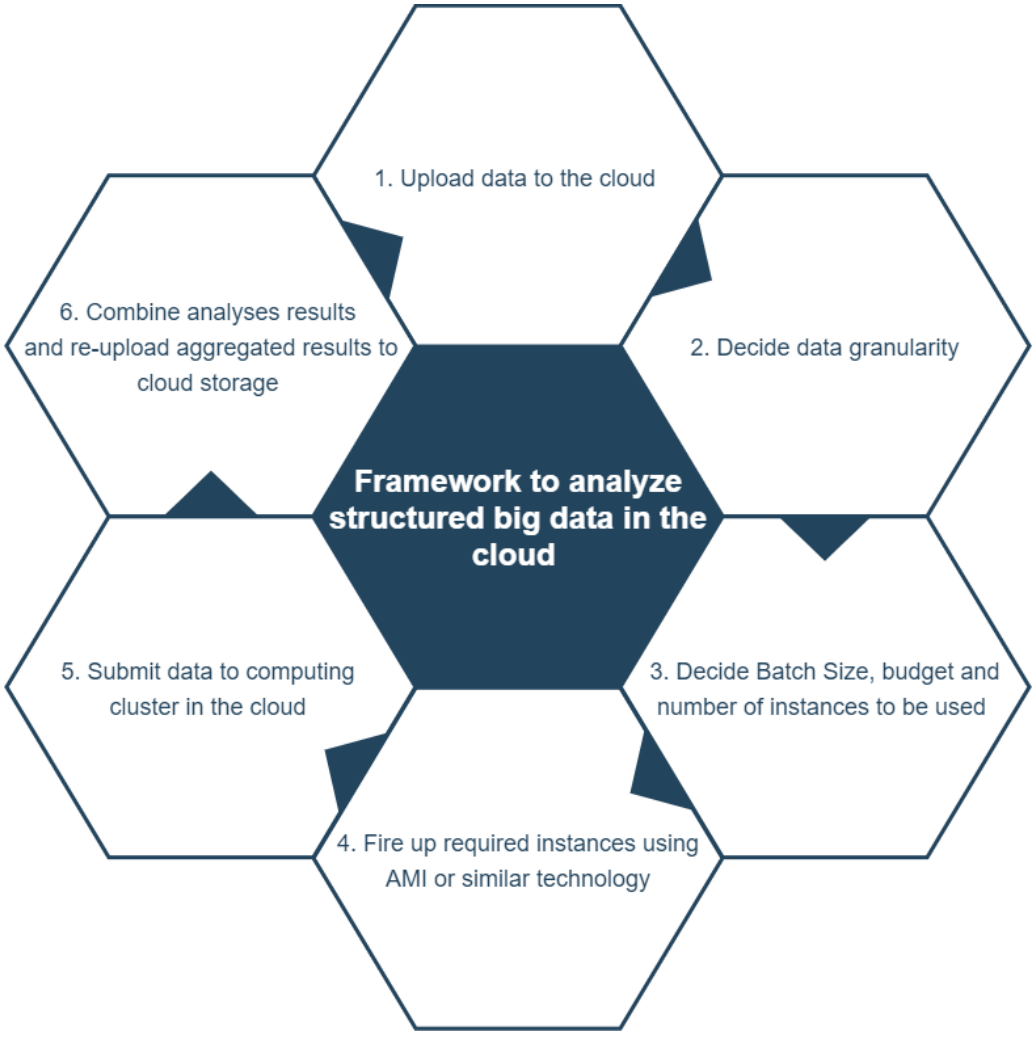


Figure 16: Framework to analyze structured big data in the cloud

5.4 Conclusion and future work

Cloud computing is a broad and vast domain. Based on this thesis' research and experiments, cloud computing can provide a stepping stone to make structured big data analysis more accessible and easier for scientists to use. Both cost and time efficiency can be ensured by using multi-node setups in spot pricing. Whereas compatibility can be ensured by using AMIs. AMIs eliminate the need to have software compatible with parallel programming and enables any software to be used in parallel on multiple machines in the cloud with virtually no issues. Scalability can be ensured by using Amazon's EBS and S3 or other similar block storage technologies. However, this thesis is not perfect, it definitely has its limitations, such as the lack of compatibility with non embarrassingly parallel programs, overlooking storage pricing, and the data size of its experiments. Nonetheless, this thesis provides a foundation for future work to make structured data analysis in the cloud even better.

Examples of future work include but are not limited to:

- Testing other services similar to AWS as they might be better or easier to use
- Compare storage pricing with other storage options as it was not taken into consideration. A good starting point would be to take a deeper look into Spark, Hadoop and AWS' storage and performance pricing.
- Test using Quantum Computing instances instead of normal CPU instances in the cloud for even better computing performance
- The possibility to build on top of this thesis' framework and try to figure out a way to fire up or shut down instances at varying stages of a certain workflow for even further cost reductions. In other words, use more or less instances in a particular workflow depending on each step's computing complexity by possibly training a ML model to do so.
- Using Machine Learning to automatically decide on granularity or sampling the data to reduce data size

References:

1. Amazon. (n.d.) *Amazon EC2 Features.* https://aws.amazon.com/ec2/features/
2. Cloud Services: S. Wagle, M. Guzek, P. Bouvry and R. Bisdorff, "An Evaluation Model for Selecting Cloud Services from Commercially Available Cloud Providers," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Vancouver, BC, Canada, 2015 pp. 107-114.
   doi: 10.1109/CloudCom.2015.94
   url: https://doi-ieeecomputersociety-org.proxy.library.uu.nl/10.1109/CloudCom.2015.94
3. Wickham, H. (2011). The split-apply-combine strategy for data analysis. Journal of statistical software, 40(1), 1-29.
4. Nick Jagiella, Dennis Rickert, Fabian J. Theis, Jan Hasenauer, Parallelization and High-Performance Computing Enables Automated Statistical Inference of Multi-scale Models, Cell Systems,Volume 4, Issue 2, 2017, Pages 194-206.e9
5. Vivek C Abraham, D.Lansing Taylor, Jeffrey R Haskins, High content screening applied to large-scale cell biology, Trends in Biotechnology, Volume 22, Issue 1, 2004, Pages 15-22
6. Davenport, T. H., Barth, P., & Bean, R. (2012). How'big data'is different.
7. Chen, L., Zhou, Y., Zhou, D., & Xue, L. (2017). Clustering enterprises into eco-industrial parks: can interfirm alliances help small and medium-sized enterprises?. Journal of cleaner production, 168, 1070-1079.
8. S. Bagga and A. Sharma, "Big Data and Its Challenges: A Review," in *2018 4th International Conference on Computing Sciences (ICCS)*, Jalandhar, India, 2018 pp. 183-187.
9. Mell, P. and Grance, T. (2011), The NIST Definition of Cloud Computing, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.800-145 (Accessed April 12, 2021)
10. Ishwarappa, J. Anuradha, A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology, Procedia Computer Science, Volume 48, 2015, Pages 319-324, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2015.04.188
11. Bao, S., Damon, S. M., Landman, B. A., & Gokhale, A. (2016). Performance management of high performance computing for medical image processing in amazon web services. In Medical imaging 2016: Pacs and imaging informatics: Next generation and innovations (Vol. 9789, p. 97890Q).
12. Oussous, A., Benjelloun, F.-Z., Lahcen, A. A., & Belfkih, S. (2018). Big data technologies: A survey. Journal of King Saud University-Computer and Information Sciences, 30 (4), 431–448.

13. Poonam S. Patil and Rajesh N. Phursule,"Survey Paper on Big Data and Hadoop Component", International Journal of Scientific and Research, Vol 3, 2014
14. O¨ zcan, F., Hoa, D., Beyer, K. S., Balmin, A., Liu, C. J., & Li, Y. (2011). Emerging trends in the enterprise data analytics: connecting hadoop and db2 warehouse. In Proceedings of the 2011 acm sigmod international conference on management of data (pp. 1161–1164).
15. Leonelli, S. (2012). Introduction: Making sense of data-driven research in the biological and biomedical sciences.
16. Williams, M. (2011). Productivity shortfalls in drug discovery: contributions from the preclinical sciences? Journal of Pharmacology and Experimental Therapeutics, 336(1), 3-8.
17. Kraljevic, S., Stambrook, P. J., & Pavelic, K. (2004). Accelerating drug discovery: Although the evolution of '-omics' methodologies is still in its infancy, both the pharmaceutical industry and patients could benefit from their implementation in the drug development process. EMBO reports, 5(9), 837-842.
18. Eddelbuettel, D., François, R., Allaire, J., Ushey, K., Kou, Q., Russel, N., ... & Bates, D. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, *40*(8), 1-18.
19. Thoman P., Jordan H., Fahringer T. (2013) Adaptive Granularity Control in Task Parallel Programs Using Multiversioning. In: Wolf F., Mohr B., an Mey D. (eds) Euro-Par 2013 Parallel Processing. Euro-Par 2013. Lecture Notes in Computer Science, vol 8097. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40047-6_19
20. Dowle M, Srinivasan A, Short T, Lianoglou S, with contributions fromSaporta R, Antonyan E (2015) data.table: Extension of data.frame. Available from:https://cran.r-project.org/web/packages/data.table/index.html (accessed 2015-12-4)
21. McCallum, E., & Weston, S. (2011). Parallel R. " O'Reilly Media, Inc.".
22. Haleem, A., Javaid, M., & Vaishya, R. (2020). Effects of COVID 19 pandemic in daily life. Current medicine research and practice.
23. Haque, A., & Pant, A. B. (2020). Efforts at COVID-19 Vaccine Development: Challenges and Successes. Vaccines, 8(4), 739.
24. Francis, R., Le Bideau, M., Jardot, P., Grimaldier, C., Raoult, D., Khalil, J. Y. B., & La Scola, B. (2021). High-speed large-scale automated isolation of SARS-CoV-2 from clinical samples using miniaturized co-culture coupled to high-content screening. Clinical Microbiology and Infection, 27(1), 128-e1.
25. McQuin C, Goodman A, Chernyshev V, Kamentsky L, Cimini BA, Karhohs KW, Doan M, Ding L, Rafelski SM, Thirstrup D, Wiegraebe W, Singh S, Becker T, Caicedo JC, Carpenter AE (2018).

CellProfiler 3.0: Next-generation image processing for biology. PLoS Biol. 16(7):e2005970 / doi. PMID: 29969450 (Research article)

26. Major J: Challenges and opportunities in high throughput screening: implications for new technologies. J Biomol Screening 1998, 3:13-17
27. Usman, S., Mehmood, R., & Katib, I. (2017, November). Big data and HPC convergence: The cutting edge and outlook. In International Conference on Smart Cities, Infrastructure, Technologies and Applications (pp. 11-26). Springer, Cham.
28. Mcdonald P, Roy A, Taylor B, Price A, Sittampalam G, Weir S, Chaguturu R. (2008) : High throughput screening in academia - Drug discovery initiatives at the University of Kansas
29. Carpenter, A.E., Jones, T.R., Lamprecht, M.R. et al. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biol 7, R100 (2006). https://doi.org/10.1186/gb-2006-7-10-r100
30. Ahmed, L., Georgiev, V., Capuccini, M. et al. Efficient iterative virtual screening with Apache Spark and conformal prediction. J Cheminform 10, 8 (2018). https://doi.org/10.1186/s13321-018-0265-z
31. Capuccini, M., Ahmed, L., Schaal, W. et al. Large-scale virtual screening on public cloud resources with Apache Spark. J Cheminform 9, 15 (2017). https://doi.org/10.1186/s13321-017-0204-4
32. Chakroun, Imen; Michiels, Nick; Wuyts, Roel (2018). [IEEE 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) - Madrid, Spain (2018.12.3-2018.12.6)] 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) - GPU-accelerated CellProfiler. , (), 321–326. doi:10.1109/BIBM.2018.8621271
33. Balachandran, B. M., & Prasad, S. (2017). Challenges and benefits of deploying big data analytics in the cloud for business intelligence. Procedia Computer Science, 112, 1112-1122.
34. Jayasree, M. (2013). Data mining: Exploring big data using hadoop and mapreduce. International Journal of Engineering Science Research-IJESR, 04 (1).
35. Zunitch, Peter (2018-01-24). "CUDA vs. OpenCL vs. OpenGL". Videomaker. Retrieved 2018-09-16.
36. Wickham, H. (2014). Tidy data. Journal of statistical software, 59(1), 1-23.
37. Dasu T, Johnson T (2003). Exploratory Data Mining and Data Cleaning. John Wiley & Sons.
38. Whitehouse, L., & Buffington, J. (2012). Amazon Web Services: Enabling Cost-Efficient Disaster Recovery Leveraging Cloud Infrastructure. Enterprise Strategy Group, White Paper.

39. FasteR! HigheR! StrongeR! – A Guide to Speeding Up R Code https://www.r-bloggers.com/2013/04/faster-higher-stonger-a-guide-to-speeding-up-r-code-for-busy-people/

Appendices

Series Vs Parallel Demonstration Using R

```
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
library(data.table)
library(ggplot2)
library(microbenchmark)

cluster <- parallel::makeCluster(4)
doParallel::registerDoParallel(cluster)

set.seed(1234)

system.time(
  rs1 <- foreach(i = seq(1000), .combine = 'c') %dopar% {  sqrt(i) }
)
```

```
##    user  system elapsed
##    0.19    0.04    0.27
```

```
parallelTest <- microbenchmark(
  foreach(i = seq(1000), .combine = 'c') %dopar% {  sqrt(i) }
)

system.time(
  rs2 <- sapply(seq(1000), function(x) sqrt(x))
)
```

```
##    user  system elapsed
##       0       0       0
```

Figure 1: R markdown file part 1

```
  sapply(seq(1000), function(x) sqrt(x))
)

identical(rs1, rs2)
```

```
## [1] TRUE
```

```
autoplot(parallelTest)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one.
```

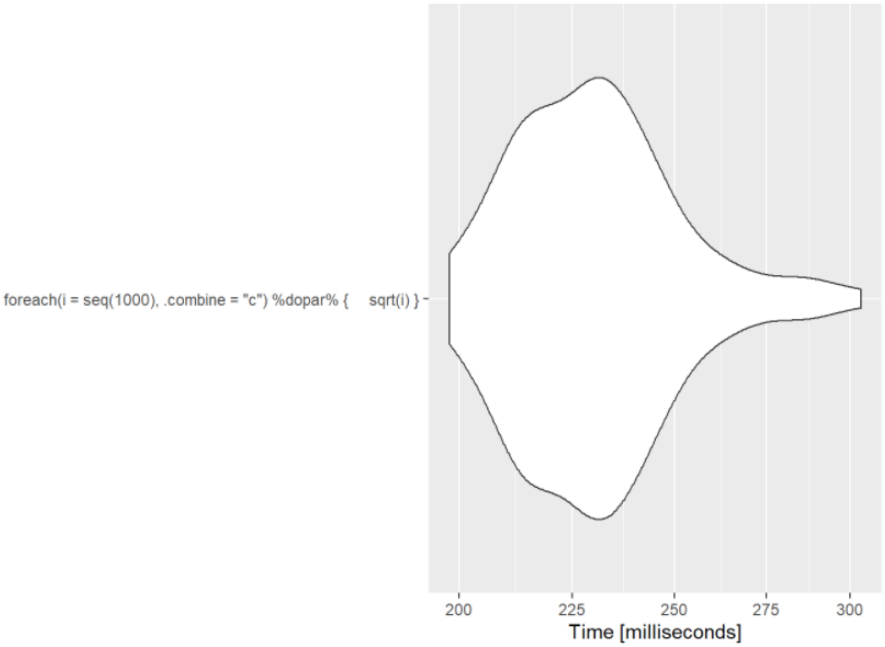
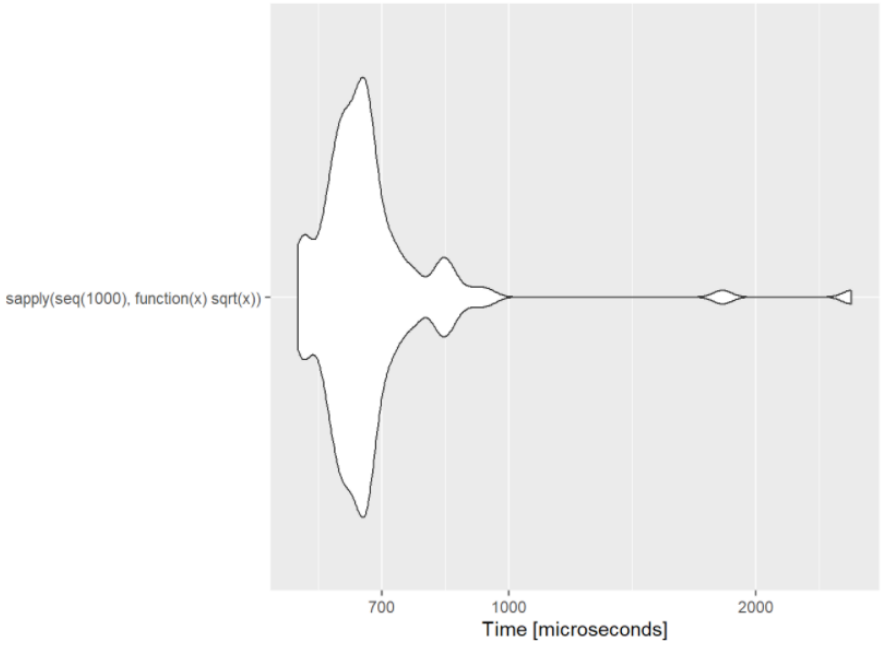


Figure 2: R markdown file part 2

```
autoplot(serialTest)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one.
```




```
stopCluster(cluster)
registerDoSEQ()
```

Figure 3: R markdown file part 3

The R markdown file above aims to compare 2 functions done in both parallel (multithreading) and series on a single machine. In situations like these where the data or the task is not very computationally heavy, it's better to do tasks in series. This is because the time it takes to partition the data and distribute it over the cores actually takes longer than the function itself.