University of Queensland

Thesis

ENGG7290: Engineering Placement Semester

Increasing the Accessibility of the Human Genome

Author
N.R. Ashford

Supervisor
Dr. Helen Huang

27 June, 2019

mychro

To all the patients, families, and loved ones affected by genetic illness.

This is for you,

I hope that one day it can help.

Summary

Genomics is a rapidly developing field, with a range of applications in clinical health care. Analysis of the human genome gives insight into the structures and variations that make us who we are. An understanding of an individual's genome can be applied to facilitate early diagnosis of monogenetic diseases (World Health Organization et al. 2002), identify risks related to particular types of cancers (Stadler et al. 2014), and customise medication so as to be most effective in treating their conditions (Lee et al. 2014).

Human genome analysis has many potential applications, but limited clinical utility. It has only recently become feasible to fully analyse the genome of a patient in regular clinical treatment (Wetterstrand 2018), and the output of such an analysis is gigabytes of hard to analyse data (Pop & Salzberg 2008). For the clinical potential of genomics to be realised, it is important that tools evolve to provide clinicians and patients with insights from the data provided by human genome analyses.

Mychro is a web based platform that streamlines the process of drawing clinical insights from human genomic data. It highlights the links between databases analysing the reference human genome (International Human Genome Sequencing Consortium 2004) and the analysed genome of a single patient, to assist in diagnosis and potential treatment of that patient's conditions.

One of the requirements of the Mychro sytem is soft real time responses to user queries. This was accomplished by storing all static genomic data in a high performance in memory index. An R tree data structure was used to show relationships between patient variations and genes found by prior genetic research.

The deployment solution of the Mychro system was designed for reliability, security, and ease of use. The system is deployed on a Kubernetes cluster, with a Prometheus server integrated into the system for monitoring purposes. Using these, an availability of 99.8% has been achieved within the Mychro system.

Protocols for inter-component communication within the Mychro system were carefully selected for ease of development and quality of the final system. The use of GraphQL has enabled automation of entire services within the Mychro architecture, and has reduced the number of network round trips needed to fulfill user queries.

A variation of Agile methodologies was used to develop the Mychro system. This approach allowed for streamlined development, with frequent reevaluation of project goals. Using Agile, user and stakeholder feedback was able to be adjusted to in a responsive manner, and changes to project requirements could be adapted to.

Contents

List of Figures

List of Tables

1 Introduction

Genomics is a rapidly developing field, with a range of applications in clinical health care. Analysis of the human genome gives insight into the structures and variations that make us who we are. An understanding of an individual's genome can be applied to facilitate early diagnosis of monogenetic diseases (World Health Organization et al. 2002), identify risks related to particular types of cancers (Stadler et al. 2014), and customise medication so as to be most effective in treating their conditions (Lee et al. 2014).

Human genome analysis has many potential applications, but limited clinical utility. It has only recently become feasible to fully analyse the genome of a patient in regular clinical treatment (Wetterstrand 2018), and the output of such an analysis is gigabytes of hard to analyse data (Pop & Salzberg 2008). For the clinical potential of genomics to be realised, it is important that tools evolve to provide clinicians and patients with insights from the data provided by human genome analyses.

Mychro is a web based platform that streamlines the process of drawing clinical insights from human genomic data. It highlights the links between databases analysing the reference human genome (International Human Genome Sequencing Consortium 2004) and the analysed genome of a single patient, to assist in diagnosis and potential treatment of that patient's conditions. This thesis discusses the components of the Mychro system, and its construction.

1.1 Overview

The rest of this thesis is presented as follows. A summary of the genomics information referenced in the remainder of the thesis is detailed in section 1.2. In section 1.3, the scope of the Mychro system as an application are discussed, as well as the goals and deliverables of the project. The completed Mychro system is discussed in chapter 2, both as a set of services and as an application.

Part I details the technical components of the mychro system. In chapter 3 the high performance genomic data store of the Mychro system is described. The data structures used, and prior art detailing optimisations to these structures, are discussed. Finally, data relating to the performance of this subsystem is analysed. Chapter 4 details the deployment of the Mychro system. The state of the art options available are evaluated, and the decisions made are justified. The protocols used within the Mychro system, both internally and externally, are discussed in chapter 5. A reflection on the effects these protocols had on development and on the Mychro system is provided.

In part II the development of the Mychro system is discussed. Chapter 6 discusses the manner in which the project was completed, from a management perspective. The approach taken is explained, and its effectiveness evaluated. The risks and opportunities

associated with the project are also discussed. Chapter 7 details the lessons learned while implementing the Mychro system. A series of reflections, each at one month intervals, are displayed. Finally, chapter 8 summarises the thesis, discussing both the technology involved and the development of the project.

1.2 Context

A genome refers to the total amount of genetic information associated with a cell, organism or species. A genome is stored as an ordered sequence of nucleotides, in long chemical chains called chromosomes (Reece et al. 2014). The human genome in particular contains 22 autosomal chromosomes, each with two copies, and then either two copies of a single sex chromosome or two unduplicated sex chromosomes. The contents of these chromosomes define everything from hair colour (Sturm 2009) through to susceptibility to certain diseases (Bluestone et al. 2010).

The behaviour of the sequence of nucleotides in any given chromosome is determined by genes, which are functional subsequences of chromosomes. The functionality of a gene is determined by the sequence of nucleotides forming it. In the context of the reference genome, this can be extrapolated from the chromosome the gene forms a subsequence of, its endpoints in the larger sequence, and whether or not it is reversed in relation to the chromosome.

An analysis of an individual genome determines the specific sequences of nucleotides that make up that individual's chromosomes. As there is only 0.01% (Consortium et al. 2015) variation between any individual's genome and the reference human genome, this data is delta encoded against the standard human genome sequence before being provided to health care workers (Li 2013). There are many types of variation from the reference genome that can be observed. This thesis investigates two clinically relevant forms of variation.

Copy number variation (CNV) A CNV is a variation in the number of copies of a chromosomal region in a genome. The length of CNVs is highly variable, ranging from a small number of nucleotides (Tassabehji et al. 1999) to the entire length of a chromosome (Lejeune & Turpin 1961). The variation in the number of copies can either be an amplification, in which there are more than the expected number of chromosomes, or a deletion, in which there are fewer.

Single nucleotide variation (SNV) An SNV is an alteration of a particular nucleotide within a genome. Alterations can take the form of additional, missing or altered nucleotides at a location within the chromosome (Strachan & Read 2011).

Measurements of variations provide clinical utility only within the context of the standard genome. The average individual's genome contains 3.5 million SNVs (Sherry et al. 2001), but the vast majority of these are benign (Landrum et al. 2013). The variations in the genome are only clinically relevant if they interfere with genes. For a CNV, this interference takes the form of an overlap between the ranges of a chromosome affected

by the CNV and the range of the chromosome containing the gene. For a SNV, interference takes place if one of the nucleotides affected by the variation is a part of that gene. International Human Genome Sequencing Consortium (2004) discovered 20-25 thousand protein folding genes in the standard human genome, each with different effects on the body. Past attempts to determine the clinical relevance of human genes have yielded several clinically relevant classifications of genes.

Kanehisa & Goto (2000) investigates the expressions of genes, and the complex biological systems formed by the relationships between gene expressions in cells. They offer a database "PATHWAY" that describes sets of genes that cooperate to perform a molecular function within cells. This database also relates sets of genes to the higher level, noticeable, effects of the functions they perform in the body.

Other research into the effects of genes on the body has produced sets of genes called panels. A panel is a set of genes that are believed to have certain traits, or exhibit certain phenotypes. Panels have clinical utility in many fields, including cardiology, oncology and inherited diseases (Illumina, Inc 2018). The genes in panels are determined experimentally, by analysing individual patients with variations affecting these genes.

At a high level, the purpose of the Mychro system is to assist in finding the genes affected by variations, and then use these databases to investigate the significance of the affected genes.

1.3 Project Goals

The purpose of the Mychro system is to make genomic data more accessible, and to ease the process of drawing clinical insight from this data. The Mychro system is implemented as a web platform that facilitates visualisation of and highlights links within the data produced during a genome analysis. The system is intended for use by clinicians to assist in genome related illnesses. The key properties of the Mychro system are defined below.

Low latency Previous prototypes leveraging existing technologies were produced in this area (appendix B) prior to the development of Mychro, but data processing latency was found to prohibit effective use of the application. To ensure that the Mychro system is usable, all user interactions are processed and responded to in soft real time. Testing has shown that users are not able to notice any latency in the application.

Reliable The Mychro system is intended for use as part of the daily work flow of clinicians. Any down time in the application will directly impact these users, and could harm the revenue of the product. At time of writing, the Mychro system has 99.8% reliability.

User friendly This tool will face experienced clinicians attempting to find links and relations in vast sets of data. The tool provides powerful insights into this data to maximise their efficiency when using it.

Secure Data from genomic analyses is highly sensitive and personal, and must be secured. All components of the system, including the environment it is deployed in, are designed for security.

The following features were considered within the scope of the Mychro system.

- Web based user interface to visualise and gain insights from patient data and the reference human genome.
- High performance application services to provide data views to the web interface.
- To scale visualisation of chromosomes, CNVs and genes.
- Reliable and secure deployment of application services.
- Extraction, transformation and loading (ETL) of data from genomic analysis.
- Interfaces with external databases of relevant genes such as "PATHWAY".

The following features were not considered to be within the scope of the Mychro system.

- Preprocessing of raw data from genomic analysis, and production of pathology reports.
- Analysis of the clinical relevance of genes beyond the relevance implied by external databases.
- Visualisation and analysis of SNVs in genomic data.

Throughout this project, the Mychro system is not the only deliverable to be produced. All deliverable artifacts of the project are detailed below.

Literature review An investigation was performed into the state of the art technology associated with the project. This investigation considered in memory indexing solutions, methods of managing and producing reliable and secure distributed systems, and protocols for use within distributed systems and web applications.

Project proposal A preliminary investigation into the viability of the project was performed. The scope and goals of the project were investigated, as well as the plans for the project and expected risks and opportunities.

Monthly reflections A series of personal reflections about the project were performed and documented. These reflections explored the learning outcomes of the project.

Interim report A report was produced that detailed the progress partway through the project. This report includes a theoretical background for all concepts involved in the project at the project at the time. Load testing was performed, and metric data collected, to facilitate a discussion of the performance of the application.

Thesis A final thesis report presents the state of the project at its completion. This thesis presents all theoretical information related to the product, and discusses the progress made in the implementation of the Mychro system. Measurements of the Mychro system were taken to facilitate analysis of its design, and reflections on the technical decisions made.

Mychro system The developed application will be taken to market at the conclusion of the project. This project delivered a deployed and available installation of the system that is ready for integration with client systems.

2 The Mychro System

The purpose of the Mychro system is to provide a user interface which allows for insights and visualisation of genomics data. To provide that data to the users, especially dynamic data such as patient records, a centralised server component of the Mychro system is required. This component also handles the more computationally intensive workloads associated with drawing insights from the genomic data. In section 2.1, the server side component of the system is discussed, and its architecture. Section 2.2 discusses the interface itself, and its components.

2.1 Architecture

At the highest level, the structure of the Mychro system is defined by its networking architecture. This architecture is defined in fig. 2.1.

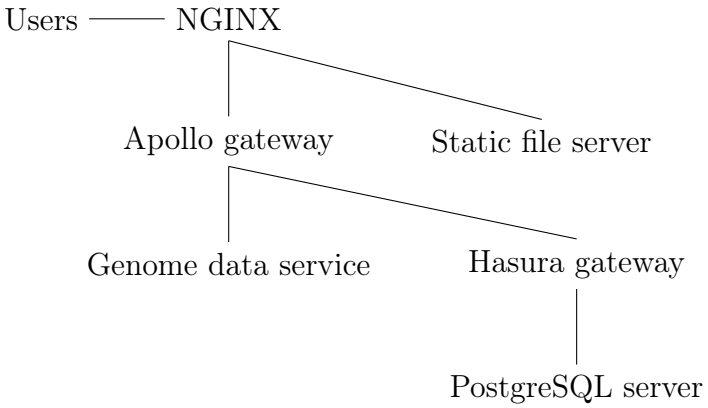

Figure 2.1: Networking architecture of Mychro system

The users are at the very edge of the networking diagram, as they have restricted access to the Mychro services. The entire user interface is a single page application, running on the user's device. This reduces the amount of network traffic required between the users and the rest of the system, as after the initial download of user interface code, only small packets of raw data are required from the servers.

The NGINX proxy is the only service accessible to the outer internet. It is responsible for TLS termination, directing requests to relevant internal services, and load balancing.

The static file server serves the user interface of the Mychro system. The proxy exposes this service at `app.mychro.io`. Where possible, the files served by this server are bundled and optimised for size.

The genome data service hosts the data relating to the reference genome. As this data

is relatively static, it is stored in a high performance, immutable, in memory index. This service exposes its data over the GraphQL protocol.

The Hasura Engine (hereafter Hasura), and PostgreSQL database, serve the data relating to the variations of individual patients. PostgreSQL is the source of truth for patient data, ensuring ACID compliant storage of this data. Hasura is configured to act as a stateless proxy over the database, transpiling GraphQL queries into SQL queries, and transforming the output of the PostgreSQL database into JSON for ease of consumption by other services.

The Apollo Gateway is a GraphQL service that performs schema stitching, combining the GraphQL endpoints offered by the genome data service and the Hasura into a single endpoint. This endpoint is exposed by the proxy at `api.app.mychro.io`.

2.2 Interface

The Mychro system streamlines the process of drawing insight from genetic data in two ways. Firstly, the system exposes the links between an individual patient's variations and study of the reference genome. Secondly, the clinical relevance of genes in the reference genome is explored. Figure 2.2 displays the high level view of a patient's genome offered by Mychro. A strong focus is placed on highlighting the variations of the patient.

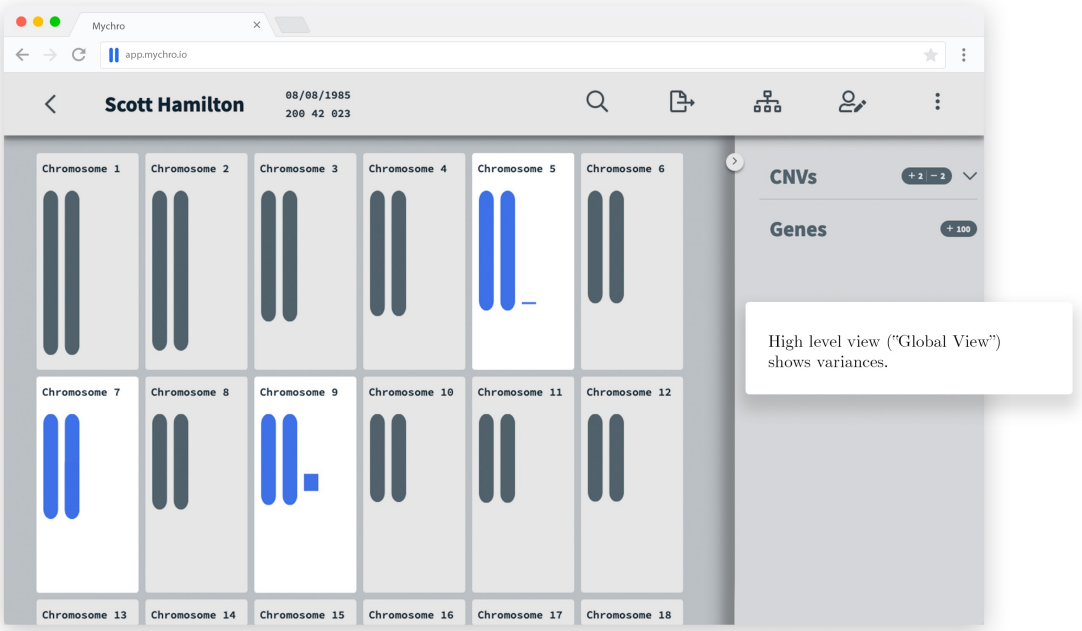

Figure 2.2: High level patient view

Due to the high variation in the size of CNVs in the human genome, not all variations

are visible in the to-scale representation of the genome that Mychro presents. To enable examinations of these variations, a sidebar presents all variations numerically, regardless of their size. This sidebar is displayed in fig. 2.3.

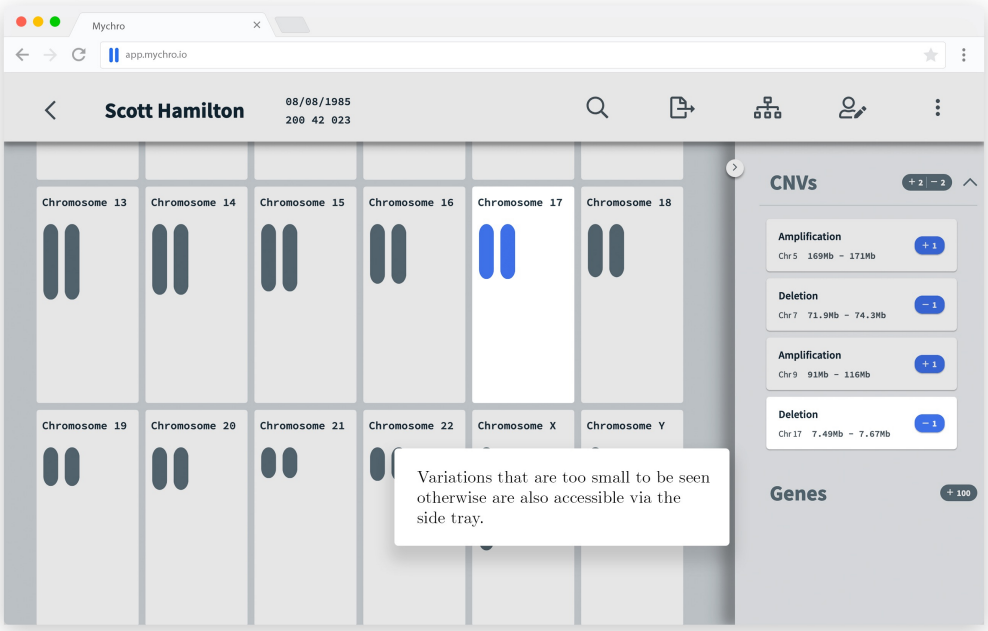

Figure 2.3: CNV representation in the sidebar

All visualisations in the Mychro application are precisely drawn to scale, highlighting the exact way in which genes and variations overlap. Figure 2.4 highlights the advantages of such a method of representation, and the insights it can yield into the nature of the relationships between variations and genes.

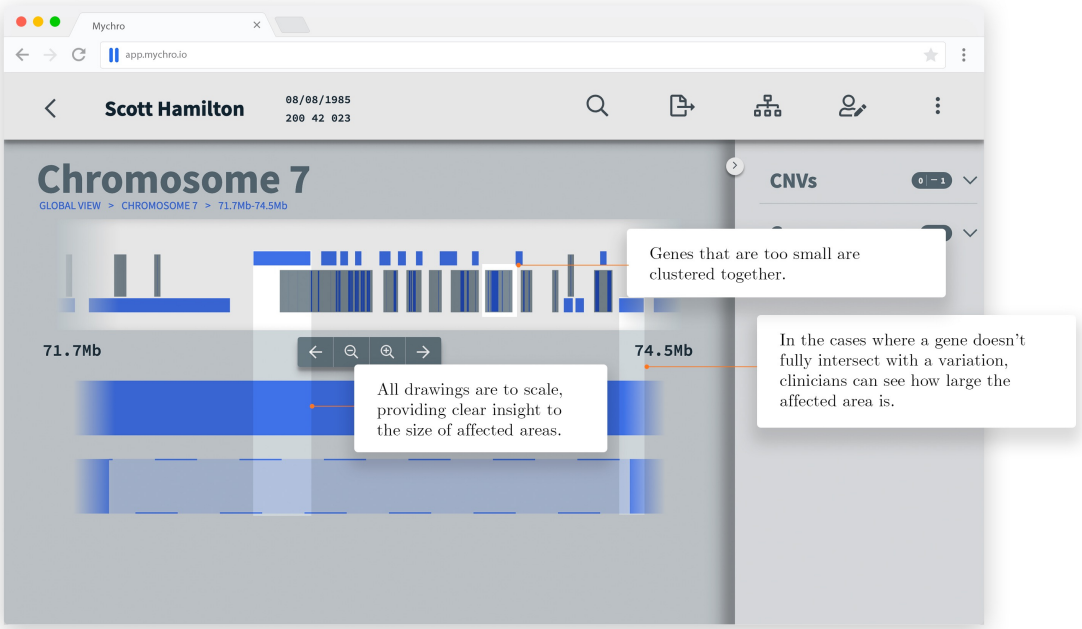

Figure 2.4: To-scale representation of genome

Gene size, similarly to CNV size, is widely varied. Within the Mychro gene view, genes that are too small to be seen or meaningfully interacted with are gathered into gene clusters. As shown in figs. 2.5 and 2.6, these clusters are fully interactive, and enable users to efficiently focus the application on the genes they represent. To further enable users to interact with small genes, the sidebar also provides information on genes when there are few enough genes to render, as seen in fig. 2.7.

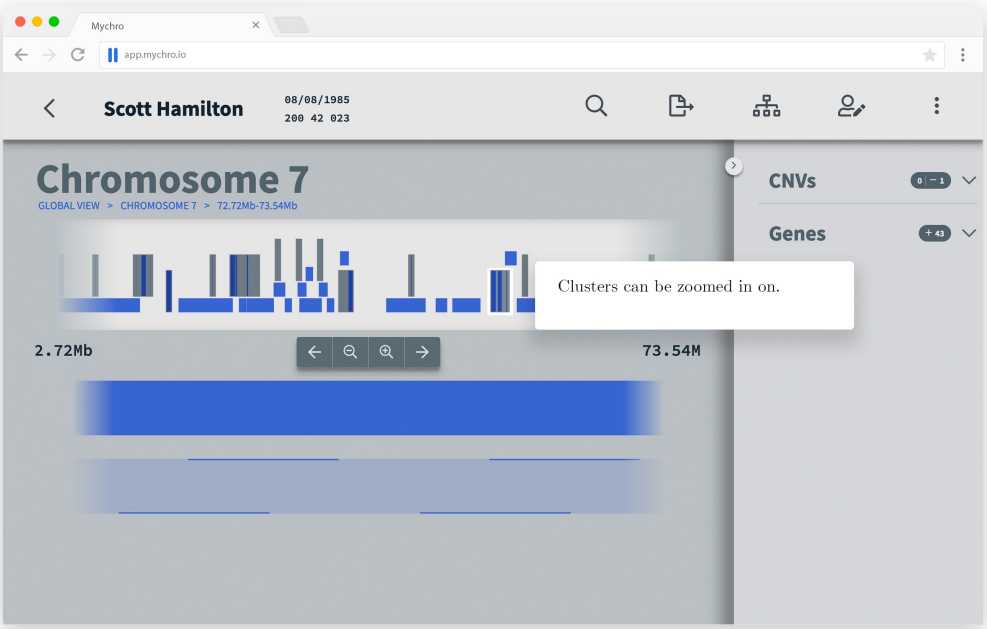

Figure 2.5: Gene clusters prior to selection

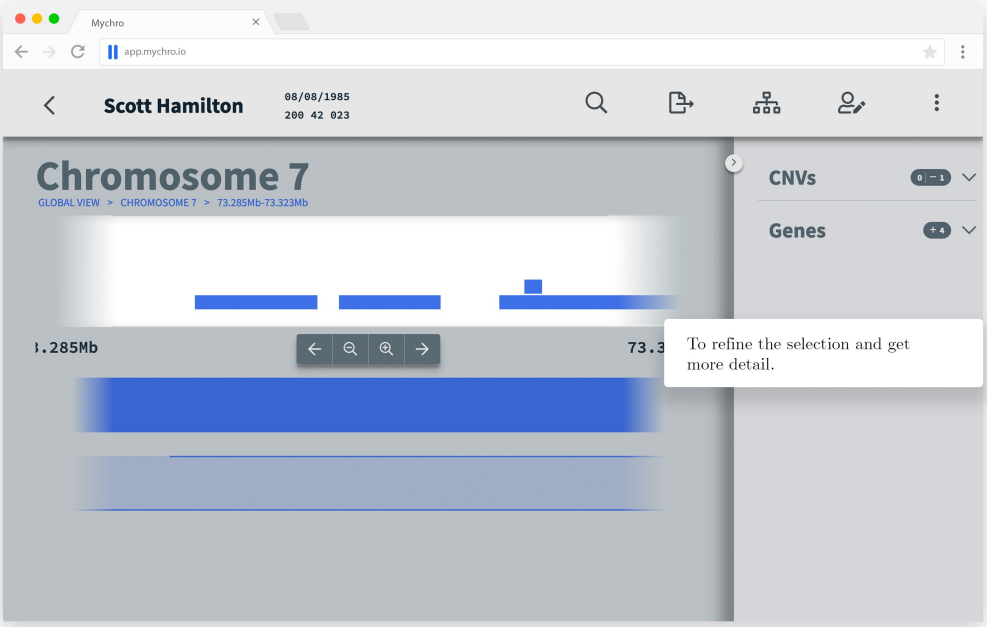

Figure 2.6: Selected gene cluster

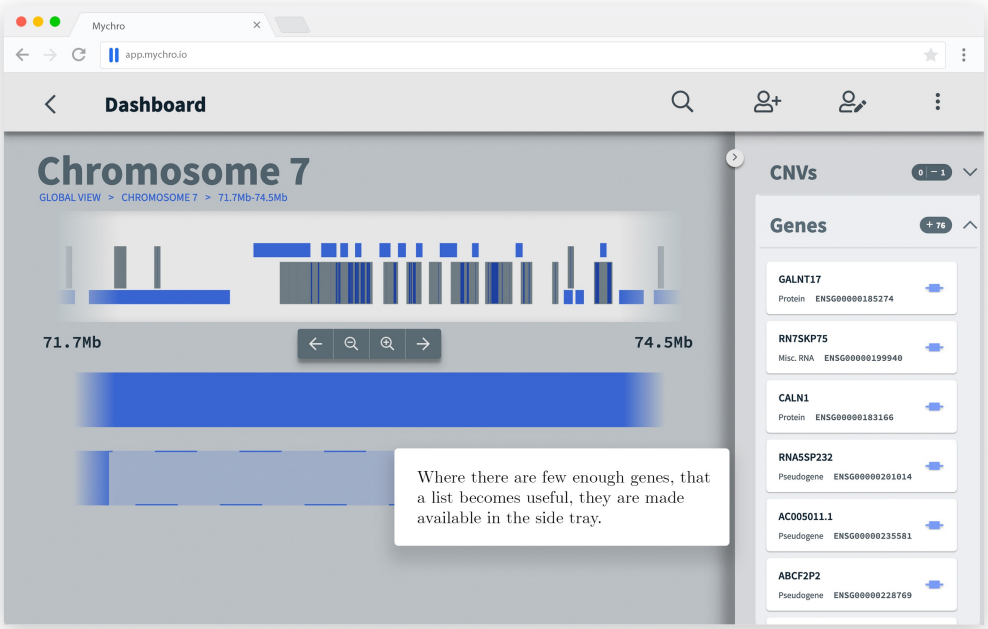

Figure 2.7: Genes in the sidebar

Information about the clinical utility of genes in the Mychro system is provided through the modals in the application. The application provides links to third party databases describing genes, as well as providing application natively, as seen in fig. 2.8.

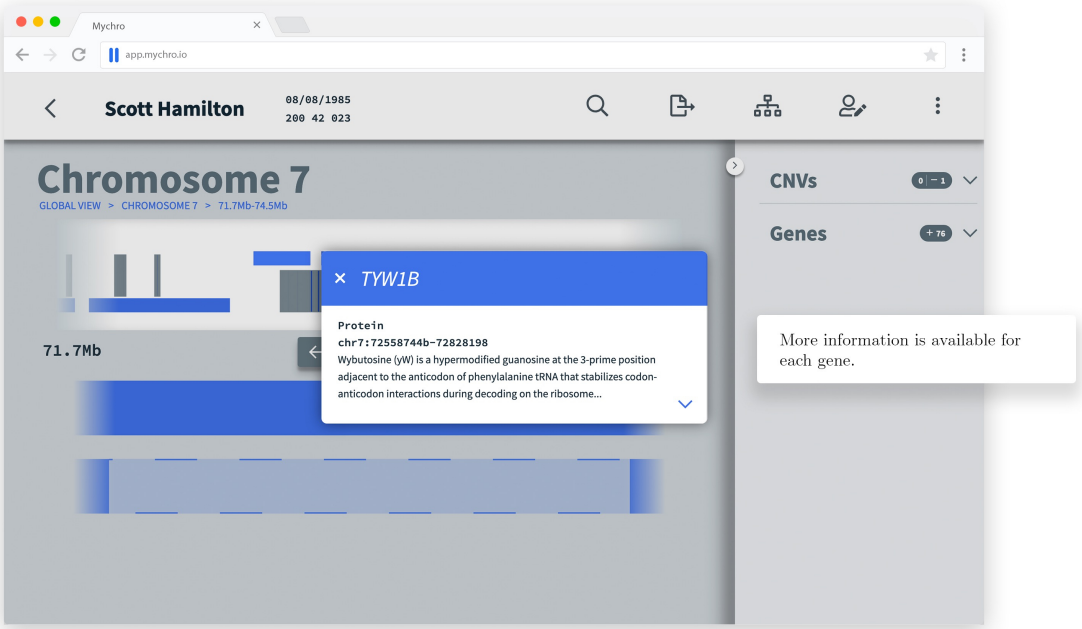

Figure 2.8: Gene modal

Part I

Technology

3 In Memory Indexing

In order to meet the soft real time requirements of the Mychro system, static data relating to the reference human genome is indexed and stored in memory. The reference human genome data used in the Mychro system can be categorised as data relating to chromosomes, and data relating to individual genes. The chromosomal data used by the Mychro system is a list of chromosomes, the number of copies they are expected to have, and their length. As there are only 24 chromosomes, the way this data is stored does not affect performance. The gene data is much more significant. There are approximately 65 000 (Reece et al. 2014) genes in the reference human genome, and though the information needed to describe a gene is very small, linking each gene to third party databases requires that several keys be associated with each gene. The purpose of the in memory index is to provide fast access to the gene data.

Within the application, gene data is used in two ways. In the visualiser and sidebar, a subset of the data available for each gene is used to provide an overview of the genes that intersect with the selection of the user. In the gene modal, an individual gene is analysed in much more detail, and all data associated with that gene is presented to the user. Selecting an individual gene by using its ID is trivial with use of a hash table (Cormen et al. 2009). However, finding the set of genes which intersect with a selected range can be computationally expensive.

Within the Mychro system, a spatial index based on a modified R tree is used to answer range based queries. Section 3.1 discusses the theory behind R trees and their application to this particular problem, while section 3.2 discusses the state of the art in R tree design. Section 3.3 discusses the performance measurements made of the current R tree in use by the application.

3.1 Theory

The problem of finding the genes contained within a selected region in the application can be considered isomorphic to the problem of filtering a set of contiguous ranges in $\mathbb{N}$ by whether they intersect a separate, given range. The selection within the application details both the chromosome being investigated, as well as the start and stop points within that chromosome being inspected. By filtering the set of genes to contain only those on the relevant chromosome, the problem is reduced to finding which genes have (start, stop) values that overlap with the (start, stop) value of the selection.

The algorithm in listing 3.1 naively attempts to find intersections between a range and a set of contiguous ranges in $\mathbb{N}$.

Listing 3.1: A naive method to find intersections

```
1 def find_intersections(set_a, b):
```

```
2     output = []
3     for a in set_a:
4       if intersects(a, b):
5         output.push(a)
6
7     return output
```

Assuming a length of n for set_a, and a length of m for output, listing 3.1 has time complexity of $O(n + m)$. While the best possible algorithm for this problem must be $\Omega(m)$, in most cases $m \ll n$ and improvements should be made on this runtime. However, without assuming or creating an index over the set, no better algorithm is possible. Any item from set_a could intersect with b, and so no item can be skipped before being checked.

Range based indexing structures on data, such as B trees, allow efficient queries of a subset of that data, based on orderings. Given a total ordering of the input data, any subset defined by elements considered greater than or less than sample points can be selected in $O(\log(n) + m)$ time. If a suitable total ordering on the set of contiguous ranges in $\mathbb{N}$ were found, this form of indexing structure could efficiently find the genes that intersect with a CNV. Such an ordering would need to be defined such that given an input range, the subset of ranges that intersect with the input could be defined by the ordering, using greater than or lesser than clauses. Appendix A shows that no such ordering exists in one dimension.

Unlike range based indexes, spatial indexing can speed up this algorithm. Spatial indexes are designed to store points in $\mathbb{R}^n$ or $\mathbb{N}^n$ space, and answer range based queries with average case logarithmic time complexity. By creating an isomorphic mapping from contiguous ranges in $\mathbb{N}$ to points in $\mathbb{N}^2$, spatial indexing can be used to speed up algorithms. Such an isomorphism is shown visually in fig. 3.1 and fig. 3.2.

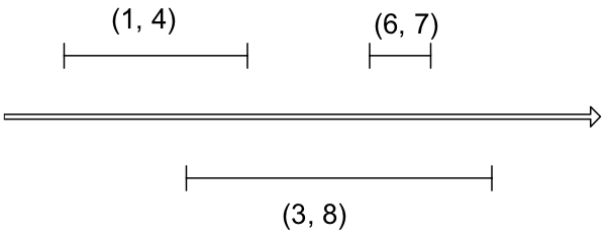

Figure 3.1: Ranges represented in $\mathbb{N}$

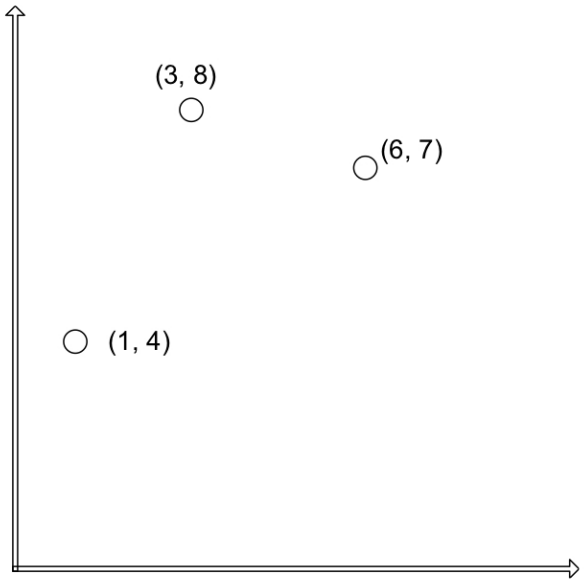

Figure 3.2: Ranges represented in $\mathbb{N}^2$

In $\mathbb{N}^2$ space, all points that intersect with range (x, y) are contained in the region $[1, y] \times [x, \infty)$. Using spatial indexing on set_b, listing 3.1 can be replaced by a range query, reducing average case time complexity of the algorithm to $O(\log n + m)$.

3.2 Literature Review

Traditional database indexing structures, such as B trees and hash maps, handle spatial data and multi dimensional range queries poorly. Modern spatial indexing solutions use variations of R trees (Guttman 1984), which are designed to efficiently handle spatial data. Similarly to B trees, R trees are optimised for secondary storage. An R tree consists of a set of nodes, each of which is described by a minimal bounding rectangle (MBR) which encompasses the data stored in that node. To reduce disk access latency, R trees maximise fan out at each later, and an individual node stores as much data as a disk page will allow.

R tree nodes are classified as leaf or branch nodes, and as root or non-root nodes. A leaf node is a block of memory that contains a set of points, and a branch node is a block of memory that contains a set of child nodes and their MBRs. When searching an R tree, the MBR of a child node is used to determine if and when that child should be fetched from disk and searched.

As nodes of an R tree are located in secondary storage, the latency of fetching them into main memory is the most significant cost in analysis of R tree operations. The cost of an R tree answering a query is measured as the number of disk operations necessary to complete the operation. For an unknown query this cost is minimised by reducing the probability of the nodes within the R tree being fetched, or reducing the probability of their MBRs intersecting the query range. Guttman (1984) shows that the area and perimeter of an MBR are related to the probability of that MBR intersecting a query

range, with reductions to these improving the performance of R trees. Beckmann et al. (1990) shows a relationship between overlap in the MBRs of sibling nodes and the number of nodes that need to be accessed to answer a query. Minimising the summed overlap between MBRs at a given layer of the tree results in better query performance for the tree.

These properties of MBRs form heuristics that describe the approximate performance of the structure. Operations on the R tree are designed to efficiently maintain low values for these heuristics, to achieve low query times. Several modifications to the R tree algorithms were found to improve these heuristic values.

Guttman (1984) provides three algorithms to insert into, remove from and update R trees. The algorithms have linear, quadratic and exponential runtime complexity with respect to the branching factor of the R tree, providing different trade offs between CPU cost and heuristic values. It was found that the slight different in heuristic values did not significantly affect query times, and that the linear algorithm resulted in the fastest overall performance.

The R* tree (Beckmann et al. 1990) was invented to reduce the CPU cost of R tree algorithms, and further optimise heuristic values. Operations on R* trees minimise overlap as well as MBR perimeter and area, and were shown to lead to faster search results. The R* tree demonstrated best case improvements to search time between 180% and 400%.

Roussopoulos & Leifker (1985) offers an algorithm for constructing an R tree from static data that produces more optimal structures. Incremental restructuring does not lead to an optimal structure, and reshuffling data within the R tree once noes have been allocated can lead to performance increases (Beckmann et al. 1990). The static R tree builds the structure from a bulk data set, optimising for low heuristics using information known about the data ahead of time. Comparisons against the original R tree (Guttman 1984) showed improvements of several orders of magnitude in heuristic values, and up to a factor of 10 improvement in the number of nodes accessed in a given search.

Traditional R trees are designed to hold data that does not fit entirely in main memory, and therefore model performance based on disk accesses. When data is able to be stored in main memory, this model breaks down. A more accurate model should consider the cost of CPU operations, and the cost of accesses to main memory when data is not in the CPU cache. With the increasing gap in latency between CPU cache access and main memory access, techniques optimised to reduce access to disk are transferable to the design of cache sensitive main memory data structures. Several attempts have been made to modify existing secondary storage focussed data structures in this way.

When porting a secondary storage data structure to main memory, the atomic element of data access changes from a page to disk size to a single cache line. A typical page size value on a modern operating system is 2^6 times larger than the cache line of a modern CPU (appendix C). To maintain the high branching factor which makes the tree structure efficient, modifications have been investigated which decrease the amount of space needed to represent each child in a node.

In the design of cache conscious B+ trees, it was found that the most significant

savings to the space required to represent a child node were in the pointer to that node (Rao & Ross 2000). On modern operating systems, the pointer to a child node in a block takes up 64 bites (8 bytes) of memory. By altering the way that nodes are stored in memory, it was possible to make significant savings in the amount of memory required. Some methods to save memory from Rao & Ross (2000) are listed below.

Aligning data types If assumptions can be made about the alignment of a block of memory, then some of the least significant bits of an address can be inferred. If a node is known to be aligned to a multiple of 2^n bytes then the bottom n bits of the address are guaranteed to be 0s. The pointer could then be stored in a $64 - n$ bit integer, and then converted with bit shifting when the pointer needs to be dereferenced.

Restricting address space If all nodes are known to reside in the same contiguous chunk of memory, with a known size and starting offset, then pointers to within that chunk can become significantly smaller. For a block of memory with size n bytes, a pointer to within that memory needs only $\lfloor \log_2 n \rfloor + 1$ bits to be stored.

Storing blocks in an array A contiguous array of length m elements, containing elements of size 2^n bytes, is able to provide guarantees about the way the memory is stored. All items in the array are guaranteed to be aligned to the size of the elements, and all items are known to reside within a contiguous block of memory $2^n \cdot m$ bytes in length. An index into such an array, rather than taking 64 bits as a pointer, takes only $\lfloor \log_2 m \rfloor + 1$ bits of memory.

Storing the children of a node in contiguous memory By storing all children of a node in a contiguous block of memory with a known order, the location of an individual child can be entirely inferred from its place within the list of children. Rather than storing one pointer per child, nodes can store only a single pointer to the start of an array. If all nodes within the tree are stored in a single array, this pointer can also be compressed using the techniques above.

In R Trees, a large usage of space is the storage of child MBRs. Compression of MBRs was found by Kim & Yun (2005) to allow for significant improvements to the amount of space required to store children of nodes deep within the R Tree. This allowed for more children to fit within a cache line, and therefore higher levels of fan out deeper within the tree. MBRs were compressed in two ways, detailed below.

Delta-encoding To reduce the range that MBR coordinates can span, each node stores the coordinates of its children's MBRs relative to the bottom corner of its own MBR. This optimisation limits MBR coordinates to fit within the bounding box of the parent node.

Bit packing If the maximum value that can inhabit an MBR coordinate is known, then the size of the type used to represent that MBR can be reduced to fit that value. For a node defined by MBR $[x, x + w] \times [y, y + h]$, only $\lfloor \log_2 (\max (w, h)) \rfloor + 1$ bits are needed to represent a given dimension in a child MBR. The total size of a child MBR with this compression method would be $4 \cdot (\lfloor \log_2 (\max (w, h)) \rfloor + 1)$.

To apply a modern spatial data indexing solution within Mychro, a variation on the R tree data structure was used. Several potential optimisations to the R tree are available, both heuristic based improvements and micro optimisations. To implement an efficient in memory spatial index, design inspiration was drawn from Roussopoulos & Leifker (1985). For future improvements, further inspiration could be drawn from Kim & Yun (2005) and Rao & Ross (2000).

3.3 Analysis

The performance of the R tree system was measured with a series of benchmarks comparing the performance of the R tree to a set of alternative indexing schemes. In each benchmark, a region of chromosome 1 was selected, and the index was used to retrieve the genes within this region. The regions were chosen to reflect common usage patterns, as determined by user testing of the Mychro system. Table 3.1 shows the ranges chosen for benchmarking.

Table 3.1: Ranges analysed in chromosome 1

Start (MB)	End (MB)	Number of genes
1	2	77
50	51	19
200	203	95

The Mychro system analyses data from the ENSEMBL (Hubbard et al. 2002) gene database. This database defines 5 317 genes within the chromosome used for testing. Figure 3.3 displays these genes as points in $\mathbb{N}^2$, as discussed in section 3.1.

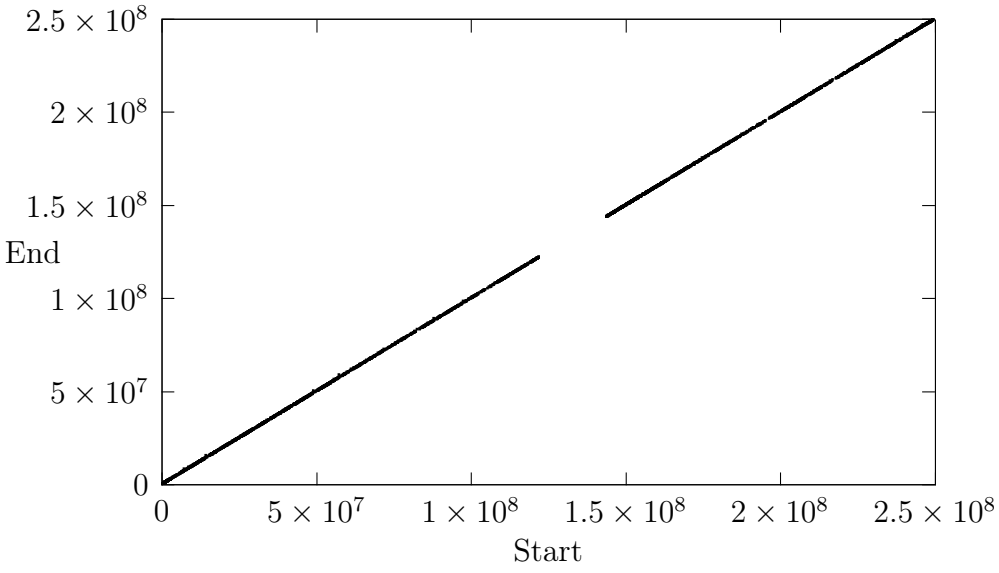

Figure 3.3: Genes of chromosome 1 in $\mathbb{N}^2$

To evaluate the performance of each indexing scheme, the time taken for the genes within each range to be retrieved was measured. To prevent excessive compiler optimisations, a checksum of the returned genes was calculated. To ensure the statistical reliability of the data, between 10^5 and 10^7 measurements were performed for each test. Outliers were removed from the data (Heisler 2018) and the mean of the remaining data was calculated. Figure 3.4 and appendix D show the results of the testing.

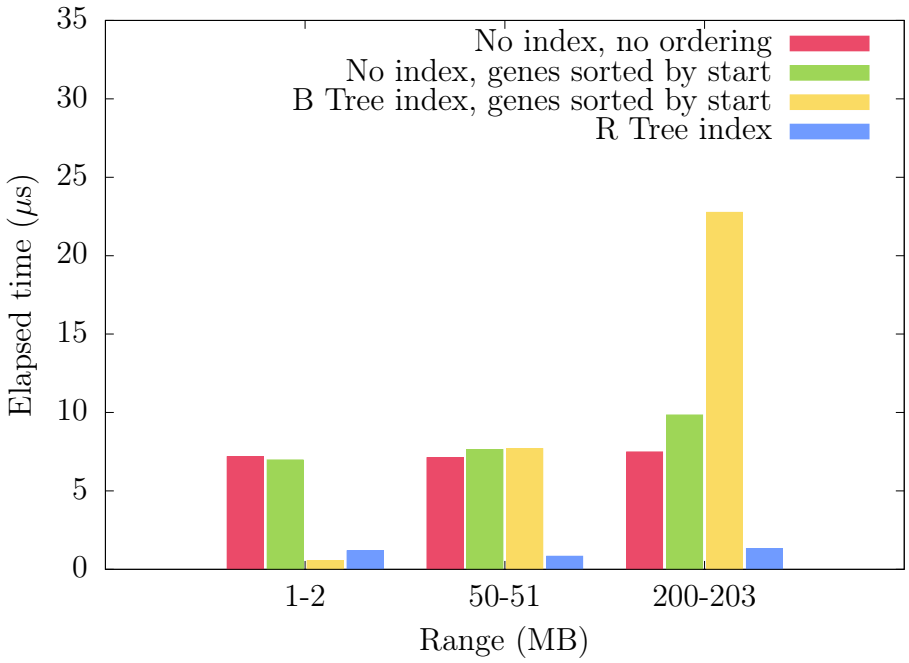

Figure 3.4: Performance of various indexing schemes

The first two test results form a baseline, showing the time taken for a brute force search and highlighting the amount of variance that can come from side channels such as CPU branch prediction and operating system scheduling. In all three cases, both of these tests were consistently around the same values.

The behaviour of the B tree in these tests highlights the effects of the ordering defined on the gene set. In these tests, the B tree index was able to filter out all genes that were too far along the chromosome to intersect, but was unable to filter out genes that were too close to the beginning of the chromosome to intersect. As can be seen, this results in a system with performance directly related to the location of the region being searched. Due to the high constant factors associated with iterating over data in a B tree, this means that in some tests the B tree index performs worse than the brute force solution. Variations, such as B+ trees, could reduce these constant factors, but the system still would not perform as well as searches through a contiguous array.

The behaviour of the R tree is consistent, and in almost all cases better than the alternative indexing schemes. In the first case, where the region is close to the beginning of the chromosome, it is evident that the R tree has worse constant factors affecting its performance than the B tree. However, in other areas of the chromosome, the superior asymptotic performance of the R tree index has a greater effect. It is also observable that the number of genes in the output set has a nontrivial effect on the performance of the R

tree, with the second range being significantly faster to analyse.

The consistently superior performance of the R tree shows that it was an appropriate data structure for the in memory index. In all tests, the R tree index responded in under 5μs. From this, and as appendix E showed that the in memory index is not a bottle neck in the Mychro system, no further optimisations to the R tree index were deemed necessary.

4 Distributed Systems

The Mychro system is deployed as a distributed system, with many interleaving components (section 2.1). Deploying such a system, in a way that is reliable and secure, is a complicated task. The requirements considered when investigating methods of application deployment are listed below.

Automated deployments To reduce the potential for human error, deployments of the system must be completely automated. In the event of an issue where manual intervention is necessary, common operations that an administrator would perform (such as rolling back to a previous release) should be automated to a single command.

Redundancy and failure resistance To achieve high availability, the Mychro system is resilient to failures. The system should support deploying redundant replicas of components, and these replicas should be isolated with respect to failure boundaries. The system should also self heal when failures are detected.

Serializible components Certain distributed system components, such as databases, require very specific operating system configuration. Components within the system need to maintain a high level of isolation from each other to prevent interference, and need to be able to run in an operating system tailored to them. To support deployment of these components, and transferring of these components between different computers, all components and their supporting run-times or operating system configurations should be able to be serialised.

Secure networking Not all components in the Mychro system should be publicly accessible. The system should protect internal components from the wider internet.

Physical hardware control Depending on the security requirements of clients, there is a possibility that the hardware that the system runs on may need to be kept on premises. The system should be able to run on user controlled hardware.

Ease of deployment To reduce development costs, a system that is easy to configure is desired. The system should also have managed hosting available on a common cloud infrastructure provider.

Ease of monitoring The Mychro system needs to support easy monitoring of resource limits, and internal application metrics such as failure rates, throughput and latency. Monitoring of these metrics is invaluable in finding and preventing production outages (Beyer et al. 2016).

The Mychro system is deployed onto a Kubernetes cluster, as this system meets the distributed systems requirements. For monitoring, a Prometheus instance is installed within the cluster. Section 4.1 discusses and compares the state of the art in containers orchestration frameworks, and justifies the use of Kubernetes within the system, while section 4.2 discusses observability in distributed systems. Section 4.3 details the behaviour of Kubernetes and Prometheus so far.

4.1 Deployment

For a distributed system to function, its components must be deployed to a set of physical machines. To ensure the system's reliability, the deployment solution should support redundant replicas and self healing. To ensure the security of the system, fine grained control over networking rules is necessary. These problems are solved by orchestration frameworks, which manage the components of a distributed system and their intercommunication.

Puliafito et al. (2019) investigates many technologies which attempt to solve these problems. The technologies of interest to this project are the orchestration frameworks, which provide a complete platform on which to build distributed systems. These are listed below.

- Amazon EC2
- Google container engine
- Microsoft azure container service
- Kubernetes
- Docker swarm
- Mesosphere marathon
- Cloudify

Due to the potential requirement for potential physical hardware control, some of these orchestration frameworks are not usable. The hosted platforms Amazon EC2, Google container engine and Microsoft azure container service cannot be run on client controlled hardware, and are therefore not applicable for the design of this system.

Orchestration frameworks have complex architectures, and are designed to run on multiple compute nodes (Puliafito et al. 2019). Setting up an orchestration framework that applications can be deployed to is a difficult task, that is drastically simplified by services offering managed installations of orchestration frameworks. Nairn (2018) indicates that the market leaders in cloud computing all offer Kubernetes as a manage service. Kubernetes is the only orchestration framework that is offered in this way, and therefore is uniquely both easy to deploy in a cloud environment, and able to be deployed on client controlled hardware.

Kubernetes natively supports features such as networking rules, self healing and through docker, serialisable components (Derstappen 2018). An ecosystem of tools has built up around Kubernetes to support features such as easy monitoring, automated deployments and rollbacks, and integration with existing third party tools (Flynn-Curran et al. 2019). For these reasons, it was decided to host the Mychro system on a Kubernetes cluster.

To achieve automation of deployments with Kubernetes, Helm was used. The entire Mychro system was described declaritively as a Helm chart, which could then be atomically operated on. Continuous integration was used to upgrade the release of the Helm chart on each commit. When manual intervention was necessary, Helm provided atomic commands to rollback releases, and to delete staging environments that were no longer needed.

4.2 Observability

While orchestration frameworks are able to increase the reliability of a distributed system through techniques such as self healing and redundant replicas, a more tightly integrated observability solution is required to detect and fix bugs or particular outages in a distributed system. By collecting more detailed information from the running Mychro application, and drawing insights from this data, developers have been enabled to detect and respond to potential application outages. The observability of this system has also enabled developers to measure the performance of the application, and perform detailed investigations into incidents, even after they occur.

Two main kinds of information can be collected from running applications: metric data or logging data. Metric data is quantifiable measurements of a running system, typically observed at regular intervals. Logging data is textual descriptions of events occurring within the system, where each individual log message is associated with an event occurred within the system (Brazil 2016). Both log messages and metric data are typically associated with timestamps, and collected into a central location for indexing, and presentation to developers. Several systems exist which can be used to aggregate logging data, metric data, or both, and produce insights from this data.

Both logging and metric data serve different purposes in a running system. Logging data is most useful for investigating past incidents in an application, by allowing developers to trace a series of events through the system. Metric data is most useful or detecting incidents as or before they occur, and assisting in responses to those incidents (Beyer et al. 2016). Another key difference between logging and metric data is the cost associated with collecting them. As logging data is associated with events, the cost associated with collecting logging data scales with the number of events occurring within a system, and therefore often scales with the amount of traffic affecting that system. In comparison, metric data is typically aggregated and is collected at a rate that is independent from system traffic. Correlation between the rate of production of observability data and the amount of system traffic can reduce the effectiveness of monitoring systems, for example by introducing confusion between a subsystem that is unable to send monitoring data and a subsystem that is not receiving traffic. For this reason, the Mychro system uses metric data for monitoring and observability.

Systems to aggregate metric data can be divided into two categories: pull based and push based systems. In pull based systems, the agent responsible for collecting metrics requests data from the subsystem being monitored; in push based systems, the subsystem being monitored sends data to the agent. Push based systems are more useful when the

system being monitored is ephemeral, such as a batch processing jobs. In other situations, pull based systems can reduce the complexity required in the system being monitored. As the components of the Mychro system are all long running services, a pull based metric system was decided on.

An investigation was performed into systems capable of performing pull based metrics aggregation. While many were considered, Prometheus was chosen for its tight integration with Kubernetes and its ease of configuration.

4.3 Findings

Through the use of Kubernetes and Prometheus, the distributed system requirements of the Mychro system have been met, and exceeded. In the 5 months the Kubernetes clusters have been running, only two incidents have affected development or the availability of the Mychro system, detailed in sections 4.3.1 and 4.3.2. The Prometheus instances have been running for 4 months, and have assisted with both development and measurements of the system.

Outside of two incidents, Kubernetes formed a solid foundation for the Mychro system. Its declarative nature allowed for ease of development, even as the architecture of the system grew more complex. Use of Helm allowed for automated deployment, as well as ease of manual interaction with the system. With the exception of a single 6 hour incident, the use of Kubernetes allowed for the Mychro system to be consistently available over a 5 month period, resulting in an availability of 99.8%. When necessary, the Kubernetes platform has allowed for 0 downtime migrations between compute nodes.

The use of Prometheus for observability in the Mychro system has been useful during the development of the project. By taking advantage of existing Prometheus integrations with external tools such as NGINX, it has been possible to measure and profile all areas of the system. Performance measurements enabled by Prometheus guided the decision to stop further optimisation of the in memory index (appendix E), and measurements of application layer errors performed by Prometheus have allowed for fast detection and diagnosis of errors.

4.3.1 Incident: Insufficient Hardware 8 May

On the 8th of May, it was discovered that the Mychro system was unavailable. An attempt was made to connect to the Prometheus server, to view current metric data and determine the cause of the unavailability, but the Prometheus server was also unavailable. Further investigation showed that all services within Kubernetes were unavailable, due to an inability to find nodes to allocate these services to. It was found that the resource requirements of the Mychro system and the associated Prometheus server, exceeded the capabilities of the three f1-micro (Google Cloud 2019) machines within the cluster.

This incident was resolved by upgrading the hardware running the Kubernetes cluster.

Three new g1-small nodes were added to the cluster, and once all workloads had been migrated to these nodes, the original nodes were terminated.

To prevent future incidents of this nature, a dashboard now actively monitors resource usage within the Kubernetes cluster. If resource usage approaches resource capacity, an alert will be sent to the development team before an outage occurs.

4.3.2 Incident: Cloud Provider Permissions Issue 5 June

On the 5th of June, it was discovered that upgrades to the Mychro system were unable to be processed. While the system was still available, continuous integration was unable to deploy new releases. From the error messages, it was deduced that the continuous integration server was unable to connect to the Kubernetes API to deploy upgrades. This was caused by a change to the permission model of the Google Cloud Platform that was hosting the Kubernetes cluster.

This incident was resolved by updating the permission settings within the new system. After this change was made, failed continuous integration jobs were restarted, and completed successfully.

As this incident did not have any adverse effects on the availability of the Mychro sytem, and was easy to resolve, it was decided not to invest in methods of preventing similar incidents from occurring in future. If such incidents become more popular, one will be investigated.

5 Protocols

The protocols selected for inter-component communication form a key part of the Mychro system. Some protocols, such as the Postgres front-end/back-end protocol (The PostgreSQL Global Development Group 2018) for communication with the PostgreSQL database, are fixed and unable to be changed. The HTTP/HTTPS (hereafter HTTP) protocol used to communicate with the front-end, and similar protocols, are fixed but allow room for extension. Other protocols, such as any communication between custom internal services, are completely unrestricted.

The selection of communication protocols has several effects on the Mychro system. As well as affecting communication overheads, and the security of the system, protocol selection affects which components can be used and which tools can be used to build custom components. The level of abstraction afforded by protocols also has significant effects on the system. A protocol for which application code can be automatically generated or more effectively reused drastically reduces the technical investment required to produce the system. Selection of protocols can also affect the complexity of the code used to interface between components.

All communication within the Mychro system, excluding that with the database, is done over the HTTP protocol. For the NGINX proxy and the static file server no extension to this protocol was necessary, but for the components relating to the API, GraphQL over HTTP was chosen for its high level of abstraction. Communication with the Prometheus metrics subsystem was performed using the Prometheus protocol over HTTP (Prometheus Authors 2019). Section 5.1 discusses communication with the front-end of the Mychro system, and the selection of application layer protocols used to communicate between custom internal services. Section 5.2 presents the results of continued development with these protocols.

5.1 API

Communication with the front-end of the Mychro system drove the selection of protocols relating to the API. Due to the requirements of browsers, this communication took place over HTTP. However, there are several areas in which protocols can be built on top of HTTP, and ways in which HTTP can be modified to suit application purposes. These areas of potential variation are listed below (Fielding et al. 1999).

- Returned content-type;
- Use of path parameter;
- Data contained in request body;

- Use of query string parameters;
- Use of cookies; and
- Use of http method verbs.

The HTTP protocol allows full parameterisation of these options, but traditionally a few common patterns are used to simplify development. These are server side rendering, REST based APIs and GraphQL.

Server side rendering is a method of using the HTTP protocol to do as much work as possible on the server side of an application. In applications using server side rendering, requests are made to the server by clicking links, or by filling in forms. Clicking a link to change the path of the browser can be used to navigate within the application and entering forms can be used either to pass data to the server or to update the application. In either case, server side rendered applications respond to HTTP requests with a content type of HTML. Whatever raw data is being presented in these applications is rendered to HTML before being passed to the client.

A REST based API is a service focussed on returning raw data, that is loosely coupled to any user interface. A REST based API is defined by its use of HTTP paths and method verbs. HTTP paths are often parameterised and each corresponds with an individual item in the store of data being served by the REST based API. For example, a user with an ID of 50 might correspond to the HTTP path `/users/50`. While HTTP paths are used to determine which item in the data is affected by a request, the effect on that data is determined by the HTTP method verb. A GET request may fetch data, while a PUT or PATCH request may update data. REST based APIs are often flat in nature and to select relational data multiple queries may be necessary. In applications using REST based APIs, connecting data together over multiple requests and rendering data to HTML are performed by the front-end.

GraphQL is a query language designed to simplify front-end development while still keeping API structure loosely coupled to the front-end application (Facebook, Inc & GraphQL Foundation 2018). GraphQL requests are parameterised entirely by a query and a set of variables, which can be contained in the request body or the query string parameters. GraphQL returns only raw data, similarly to REST based APIs, but connecting of linked data is performed on the server side. GraphQL is designed so that all related data needed in a request can be returned by a network round trip. Constructing these queries and rendering the returned data to HTML is performed by the front-end in applications using GraphQL.

These three options were evaluated on their fitness for purpose using several criteria. The primary criteria was effects on the application, in terms of performance and user experience. Other criteria included complexity of front-end code, complexity of back-end code, and potential for future expansion.

The choice between these three variations on HTTP could have numerous effects on the final Mychro system. Ideally, the system should only require one network round trip to transmit data, as multiple round trips can increase latency. To improve user experience,

all loading should be done asynchronously, with the rest of the application still usable during loading. REST based APIs require multiple network round trips to fetch data, while server side rendering prevents asynchronous loading of data. Use of GraphQL does not have any adverse effects on the application.

Complexity is added to the front-end of the application by introducing the need for the front-end to render content to HTML and by introducing the need for the front-end to perform network requests. Of the three protocol options, server side rendering adds very low amounts of front-end complexity, while GraphQL and REST based APIs add nontrivial amounts. Due to the need to string multiple network requests together, REST based APIs increase the complexity of front-end development significantly more than GraphQL. Due to the detailed specification of GraphQL, libraries are able to provide more comprehensive support.

Complexity in the back-end of the application depends on the work being done by the back-end and on the support of client side libraries. In the case of both GraphQL and REST based APIs, tools such as the Hasura Engine (Hasura, Inc 2016) and GraphCool Prisma (Burk 2019) exist to automate the process of creating a standalone server entirely. In these cases, back-end complexity is drastically reduced. In the case of GraphQL in particular, tools such as Apollo (Hasura, Inc 2018) can also automate the process of combining multiple servers to a single endpoint, allowing for a combination of custom application logic and automated services. For server side rendering however, back-end complexity is high. Library support is not as powerful and the complexity of rendering data to HTML must be shifted to the back-end also.

From evaluations of these criteria, especially the primary criteria of application functionality, GraphQL was selected as the most appropriate protocol for data transfer to the front-end over HTTP. This influenced the selection of back-end services such as the Hasura Engine and an Apollo Gateway to connect Hasura to the in memory index.

5.2 Results

The protocols selected for communication within the Mychro system have enabled faster development and a more optimal application. The use of GraphQL has allowed for automation of significant components of the application and has supported streamlining of data transfers. Use of the Prometheus protocol has reduced the cost of metric aggregation and permitted more fine grained measurements without prohibitive overhead.

Automation of development within the Mychro system through the use of GraphQL has affected three areas of the application. Primarily, all interaction with the database has been automated through the Hasura Engine, a service which transpiles GraphQL queries directly to SQL queries. From the introspection capabilities of GraphQL, it was possible to install an Apollo gateway which seamlessly combines the two back-end APIs to a single endpoint. Finally, within the front-end, Apollo client side libraries were used to simplify relationships between the data rendered in components and the data from back-end APIs. These high levels of abstraction and automation have streamlined the development process as well as increased developer productivity.

Use of the GraphQL protocol has also lead to performance improvements within the Mychro system. The SQL queries produced by the Hasura gateway allow for all data to be collected from the database in a single network round trip. Caching within the Apollo client side libraries and the Apollo gateway has allowed for reduced network traffic and latency.

The optimisations of the Prometheus protocol have improved the quality of metric data aggregation within the Mychro system. By sending accumulated data to the Prometheus server, metric aggregation has become resilient to dropped network packets and the level of detail in monitoring data can be altered dynamically. Customisation of parameters within the protocol has allowed for more precision in metric data where it is necessary, without collecting in depth data unnecessarily.

Part II

Development

6 Project Methodology

Work on the Mychro system was completed using a variation of agile methodologies. With a small team, over the course of six months, the project was completed in a flexible manner, with adjustments being made to the goals of the project as necessary. Section 6.1 discusses the approach to the project, including where automation was included. The timeline of the project and key milestones are detailed in section 6.2. The remainder of the chapter discusses the resources used by the project, and the risks faced.

6.1 Approach

Work on the Mychro system was completed using a variant of agile methodologies All work was completed in two week sprints, interspersed with sessions of planning, retrospection and evalution. A sprint is a commitment to complete exactly two weeks of uninterrupted work, based on the assumptions about the project state made at the beginning of the sprint. In between sprints, these assumptions are challenged and reevaluated, and adjustments are made to the project plan for future sprints. Frequent reevaluation of the project and its goals ensured that the project trajectory remained relevant to the needs of the stakeholders, and avoided the issues associated with large amounts of ahead of time requirements collection.

Each sprint began with a planning session. During this session, the backlog of work was reevaluated. Items within the backlog were sorted by their priority, and by their dependencies. Using estimations of the complexity of the items at the top of the backlog, the scope of the sprint was defined as a set of tasks from the top of the backlog that could be completed in two weeks. At the end of the two weeks, the sprint ended, and all uncompleted tasks from the scope of the sprint were returned to the backlog. The deadline for the planned end of the sprint was prioritised above the completion of the sprint's work to maintain the balance between committed work periods and frequent reevaluation. If all planned tasks were completed before the planned end of the sprint, extra tasks from the top of the backlog would be added to the scope of the sprint.

The conclusion of a sprint was marked by three events. All work completed during the sprint was deployed to the production environment, the team showcased completed work items from that sprint to the stakeholders, and a retrospective session was held. Within the retrospective session, team members and stakeholders reflected on the completed sprints, discussing which had strategies had worked well and which had worked poorly, and considering new approaches to take in future sprints. These retrospective sessions shaped the approach to work in future sprints.

Work within sprints was divided into tasks, each with a defined scope and clear definition of completion. Each task was tracked by the project management tool Jira, and assigned an ID. Work on a task was completed in a separate branch, named for the Jira

ID, in the source control management tool git. Git served as a single source of truth for all work on the project, including the LaTeX source of the reports and the state of the deployed production system (described by branch `master`) and staging environments (each associated with a separate git branch). Work on a task during a sprint could be described by the motion of the associated Jira task through five stages of completion, and by the associated git branch. These stages are detailed in table 6.1.

Table 6.1: Stages of work items

Stage	Description
To do	Tasks in this stage were on hold, to be attempted when other items in the sprint were completed. This stage also described tasks that had not yet been assigned to a sprint. Tasks in the To do stage did not have associated git branches.
In progress	Tasks that were in progress were currently being completed. When the task entered this stage, a git branch was created from the active development branch `dev`. The Gitlab CI tool consistently validated all code pushed to the git branches associated with in progress tasks, and maintained a staging environment that was kept up to date with the source code on that branch.
Review	Before the git branch associated with an in progress task was merged back into `dev`, the work completed for the task went through a strict review process. A separate team member would review the quality of the code, and perform manual testing on the staging environment for that branch, before the task was considered ready to be merged.
Quality assurange	Before the end of a sprint, a final check was performed on all tasks completed during the sprint to ensure that the project still functions. This test would be performed on the staging environment associated with the `dev` branch.
Done	Tasks in this stage were completed.

Use of Agile methodologies facilitated a smoother development process on the Mychro system. Feedback from users and other stakeholders could be quickly responded to, as could changes to project requirements.

6.2 Timeline

The key milestones and the timeline of the Mychro project are best defined by the high goals of each sprint in the project. These are detailed below, along with the start and end dates of the sprints.

Sprint 1 **9 January - 22 January**

- Configuration of deployment and staging environments.
- Configuration of continuous integration, and deployment.
- Production of low fidelity prototypes of the user flow within the application.

Sprint 2 **22 January - 5 February**

- Implementation of high level visualisations of static data within the application.
- Production of a low level design system to be used within the application.
- Implementation of monitoring deployed environments.

Sprint 3 **5 February - 19 February**

- Completion of online reflective journal 1.
- Production of branding material for the project.
- Implementation of basic zoom features within the application.

Sprint 4 **19 February - 5 March**

- Completion of project proposal draft.
- Production of high fidelity prototypes of the application.
- Adjustments to the application, to match the prototypes.

Sprint 5 **5 March - 19 March**

- Completion of project proposal, and online reflective journal 2.
- Implementation of low level gene view within the application.

Sprint 6 **19 March - 2 April**

- Implementation of a spatial query data structure to speed up the application.
- Implementation of a side menu for the application.

Sprint 7 **2 April - 16 April**

- Completion of online reflective journal 3.
- Completion of interim report draft.

Sprint 8 **16 April - 7 May**

- Completion of interim report.
- Response to feedback from user testing.
- Implementation of gene timeline, and gene clusters.

Sprint 9 **7 May - 21 May**

- Completion of online reflective journal 4.
- Implementation of gene explosion algorithms.

Sprint 10 **21 May - 4 June**

- Completion of part of draft of final thesis.
- Implementation of gene modal.
- Integration with database, and implementation of multiple patient system.

Sprint 11 **4 June - 18 June**

- Completion of online reflective journal 5.
- Completion of draft of final thesis.
- Completion of oral presentation.

Sprint 12 **18 June - 27 June**

- Completion of final thesis.

6.3 Resources

Several external and internal resources were required to produce the deliverables of this project. These are detailed below.

The wider academic community have already attempted to solve a number of the problems that were faced in this project. Research papers, journal articles and books available through the UQ library in both computer science and genomics were used to produce the literature review, and to influence project design. As well as these, white papers and other publications by development teams within the technology industry were referenced.

Many external software libraries and subsystems were used in the development of the Mychro system. The source code and compiled outputs of these systems was vital to the deployed application. The technical documentation and source code of these systems was referred to when interfacing with them.

For both development and production environments, virtual CPUs hosted by a cloud provider were used to host the Mychro system. Cloud computing platforms removed the need for manual administration, acquisition and maintenance of hardware, and offer high levels of availability with service level agreements. Cloud computing also facilitated automated scaling of the project, through dynamic computing resource allocation.

A small team of Aginic employees, key stakeholders and academics also supported this project and the associated thesis. The members of this team are listed in table 6.2.

Table 6.2: Team members

Name	Role
Rahul Nair	Delivery lead.
Jason McLaren	User interface and user experience designer.
Alan Robertson	Product owner and geneticist.
Dr Helen Huang	Academic supervisor.
Dr Michael Gabbett	User representative (clinical geneticist).
Dr Robert McLeay	User representative (computational biologist).

6.4 Risks

To better analyse the risks to the Mychro project, a quantitative risk assessment matrix was used. The likelihood of an event occurring, and its consequences, were analysed using table 6.3 to determine the severity of the risk associated with that event.

Table 6.3: Risk matrix, consequences vs likelihood (Carmichael 2017)

	Insignificant	Minor	Moderate	Major	Critical
Almost certain	Medium	Medium	High	Extreme	Extreme
Likely	Low	Medium	High	High	Extreme
Possible	Low	Low	Medium	High	Extreme
Unlikely	Low	Low	Medium	Medium	High
Rare	Low	Low	Low	Medium	High

The likelihood levels were defined by the expected frequency of the event, under normal working conditions. The levels are defined in table 6.4.

Table 6.4: Risk likelihoods (Carmichael 2017)

Likelihood	Expected frequency
Almost certain	Once per day.
Likely	Once per week.
Possible	Once per month.
Unlikely	Once per year.
Rare	Once every five years.

The severity of a risk determined whether it was acceptable for the project to continue without the risk being mitigated. The levels of severity, and their acceptability, are defined in table 6.5.

Table 6.5: Risk severities (Carmichael 2017)

Severity	Acceptability
Extreme	Not acceptable.
High	Not acceptable, except in very rare circumstances.
Medium	Broadly acceptable.
Low	Acceptable.

The risks affecting the Mychro project can be divided into safety risks and business risks, where each are analysed separately. Safety risks are potential events which could affect the safety and well-being of team members or others. The consequence levels for safety risks are defined in table 6.6. The safety risks themselves are listen in table 6.7.

Table 6.6: Safety risk consequence levels (Carmichael 2017)

Consequence level	Definition
Critical	Fatality / multiple fatalities, or permanent impairment.
Major	A serious injury / illness, with potential for temporary impairment.
Moderate	A moderate injury / illness, with potential for reversible impairment.
Minor	First aid is required, or equivalent.
Insignificant	No injury or illness.

Table 6.7: Safety risks

Risk	Consequence	Likelihood	Severity
Repetitive strain injury from keyboard use	Moderate	Rare	Low
Eye strain from prolonged screen exposure	Moderate	Rare	Low

Both of these safety risks have low severity, and were therefore deemed acceptable. The risks were managed by ensuring that regular breaks were taken from computer work.

Business risks are potential events which could affect the successful completion of the project, or the success of Mychro as a product. The meaning of consequence levels for business risks is detailed in table 6.8. The business risks that could occur over the course of the project are listed in table 6.9.

Table 6.8: Business risk consequence levels

Consequence level	Definition
Critical	More than one month of work on the project is lost, or the project cannot be completed.
Major	One month of work on the project is lost.
Moderate	One week of work on the project is lost.
Minor	One to two days of work on the project are lost.
Insignificant	No affect to the completion of the project.

Table 6.9: Business Risks

Risk	Consequence	Likelihood	Severity
Illness in a team member	Minor	Possible	Low
Test users withdraw support for the project	Moderate	Unlikely	Medium
Requiring certification to access patient data	**Critical**	**Unlikely**	**High**
Major outage in cloud software provider	Minor	Rare	Low

The risk of losing significant amounts of time to gain certification of the application was unacceptable. This risk was mitigated, and removed from the project, by ensuring that the application itself does not generate any medical data or offer any medical knowledge. The Therapeutic Goods Administration of Australia reviewed the concept of the Mychro system, and currently does not consider it a medial device. Instead, the Mychro system was classified as a marginal product. To prevent any changes to this classification the Mychro team has an open channel of communication with the TGA and have designed Mychro to meet the requireents of a Class IIa medical device.

The risk of losing test users had medium severity, and was mitigated by finding a secondary user from within the target user demographic who could temporarily perform user testing. Alan Robertson (table 6.2) is a geneticist, and was able to fill this role if needed. The consequences of the risk were therefore lowered.

The amended business risks that could occur over the course of the project are presented in table 6.10.

Table 6.10: Ammended business risks

Risk	Consequence	Likelihood	Severity
Illness in a team member	Minor	Possible	Low
Test users withdraw support for the project	Minor	Unlikely	Low
Major outage in cloud software provider	Minor	Rare	Low

All business risks were low severity, and therefore acceptable.

7 Professional Development

Work on the Mychro system was completed in a professional engineering environment, at Aginic. Several lessons were learned working in this environment, and these are displayed in the form of personal reflections in this chapter.

7.1 Configuring Monitoring (February)

For unknown reasons, I was experiencing issues with the uptime not only of my application, but also of some of the underlying infrastructure. After sorting priorities and allocating time, I spend three to four days researching and implementing solutions to dynamically monitor the state of the applications within the Kubernetes cluster. I investigated, and attempted to implement, three different solutions before arriving at one which worked - a Prometheus installation on the cluster and manual aggregation of Prometheus style metrics within my application. By the end of it, I had discovered that the reason I was experiencing issues was that I did not have enough RAM available in the cluster, and so I reconfigured the nodes and it started working again. In future, if things start to have issues, I now will have access to a time series database full of metric data to use while debugging.

Configuring monitoring was the first hard issue I've had to solve, without a clear solution, in this project. I was surprised and somewhat disappointed at the state of the ecosystem surrounding monitoring of distributed systems in Kubernetes, with most tools being very difficult to successfully configure. By the time I had figured out a solution I had more of an appreciation of the amount of complexity in trying to solve metric aggregation in a distributed system, especially in issues relating to reliability. The metrics servers were all designed to be happily functioning even when the rest of the services are down, so that you have something to tell you why the services are down, and the solutions to that style of problem are impressive to integrate with and study.

This was a hard problem to deal with, and I worried that some of the solutions I had come up with were very over engineered. I discussed my ideas about the problem, and my ideas for solution, with my coworkers regularly - in particular the senior engineers. This gave me some outside opinions and useful perspectives on the problems I was solving, even if the ideas they suggested did not end up working.

I gained an appreciation of monitoring and metric aggregation in deployed software systems, and I learned to grasp the significance of "flying blind" and how drastically it can hinder operations work in repairing failed services. In university, we never touched on logging or metrics, and so learning not only that this sort of information can be gathered, but also the best practices in how to gather and store the information in complex systems was entirely new ground for me. In future I will not wait until things start to break to start investigating what is or could be going wrong. Instead I will place a higher priority

on gathering information from the start, so that fixing problems is easier and so that hopefully I can pro-actively counter issues before they lead to total outages.

The following EA stage one competencies were developed during this event.

EA 1.1 Application of established engineering methods to complex engineering problem solving.

7.2 Estimation Issues (March)

When the project proposal assessment details were released, I decided to start it immediately to get a draft to my supervisor sooner. At the beginning of the next sprint, I created a work item to complete the draft to the project proposal, and added it to the upcoming sprint. It was estimated to be of medium complexity, and several other items were added to the sprint to be attempted after the draft was completed.

Unfortunately, this estimate was inaccurate. Rather than taking the few days that I expected, completing the draft took a week and a half of the two week sprint. As a result of this, the majority of the other work items I had planned for the sprint were not completed.

At the showcase and retrospective at the end of the sprint, the rest of the team and I reflected on why nothing had happened in the sprint. My supervisor and the other project stakeholders were okay with nothing having been completed, but would prefer that the estimates were more accurate so that we would know at the start of the sprint. As estimation is an imprecise art, there were no consequences to the estimate being inaccurate.

Despite not having completed the work items planned for the sprint, the sprint ended on time after 2 weeks. In the retrospection and planning for the next sprint, we reevaluated the priorities of the backlog. Some of the tasks that were not completed are now no longer considered priorities. In future estimation of report writing tasks, my supervisor and I have agreed to increase our estimates of the complexity.

I gained an appreciation of how agile methodologies can adapt to poor estimation and requirements gathering. Estimating the complexity of tasks that have not been attempted is error prone, as is ahead of time requirements gathering. Having a project management framework that is able to adapt to issues caused by these issues was really useful, and I look forward to applying it both in the rest of this project and in my career.

The following EA stage one competencies were developed during this event.

EA 2.4 Application of systematic approaches to the conduct and management of engineering projects

EA 3.5 Orderly management of self, and professional conduct

7.3 Overestimation of performance requirements (April)

One of the main problems anticipated in this project was soft real time performance when dealing with large amounts of genomic data. I have done a lot of research into complex methods of indexing this data. Recently in the project, I reached the stage where I needed to ingest one of the largest data sources (the human gene list) into my application and index it for searching. I implemented a simplified version of the indexing data structures I had been researching, with the intention to later revisit the code and apply optimisations. In testing, I found that this unoptimised data structure had no issue responding to queries in a timely manner.

This discovery had several effects. Firstly, due to overestimations we now have more time available to complete the rest of the project that we did not anticipate, and this could allow for an increased scope of the product. Secondly, as I will not need to be look in depth at ways to tune the performance of the data structures involved, I will have less to write about in my thesis report. Finally, some other performance based decisions made in the past (such as the choice to write the body of the software in rust) have been placed in question.

The only action currently being taken as a result of overestimation of the indexing task is that another task was introduced to the scope of that sprint. Aside from that, the project is continuing as usual, and we continue to evaluate the scope of the project and highest priority work items at the end of each sprint. In the thesis report, I intend to discuss alternative avenues of investigation with my supervisor, such as the algorithmic improvements allowed by the data structure in use rather than low level optimisations of it. Due to the amount of work that has already taken place under the assumption that speed would be necessary for the application, I have no plans to rewrite the software in a less performance sensitive language.

I have learned from this never to make premature assumptions about the requirements of a product. Attempts to optimise the project before testing to see that that optimisation was necessary has resulted in a few design decisions that could have been made differently had the performance requirements been more accurately known beforehand. In future, I will aim to be more agile in my approach to projects, and not invest effort into meeting requirements that have not been validated.

The following EA stage one competencies were developed during this event.

EA 2.3 Application of systematic engineering synthesis and design processes.

7.4 Overextension of spike (May)

The designer and I were asked to perform a time boxed spike, spending 3-5 hours attempting to implement animation within the product. If the work could not be completed within that time frame, we were to stop and attempt the feature at a later date when it could

be prioritised higher. After 2 hours we had implemented the feature, but the code was of a low quality. Impressed with our work, we decided that we would take a short period of time tidying the code and then surprise the product owner with a fully implemented animation feature at the end of the sprint. The code took longer than anticipated to complete, and so by the end of the sprint we had spent more time than we should have, and kept the product owner in the dark as to what we were doing.

The lack of visibility worried both the product manager and other stakeholders in the project. While everyone was okay with us spending time on the feature, given the results it produced, the lack of visibility into what we were doing was frowned upon. Thankfully no long term effects were felt.

In future, we will not attempt to surprise members of the team with what we are doing. Visibility within the team will be emphasised as a goal of the agile process, and all team members will be encouraged to get each other back on track if something similar happens again.

I have learned the magnitude of the effects of poor visibility into how team members are spending their time. I was not previously aware of the consequences of this, and how it would affect not only team members but also external stakeholders trying to monitor development of the project.

The following EA stage one competencies were developed during this event.

2.4 Application of systematic approaches to the conduct and management of engineering projects.

3.1 Ethical conduct and professional accountability.

3.4 Professional use and management of information.

3.5 Orderly management of self, and professional conduct.

3.6 Effective team membership and team leadership.

7.5 Code Review (June)

I was working on a feature to add to the application, on a separate git branch. Something came up that blocked my work on the feature for a few days, so as I went I continued adding other small changes and fixes to the application - on the same git branch. By the time I was ready to complete the initial feature and attempt to merge the branch back into the trunk branch, there were far too many changes for an effective code review.

The developer assigned to the code review told me as soon as they saw it that the review was too large. I was instructed to go back and split it up into several smaller git branches that could each be reviewed independently so that the reviewers could still look at the code in detail.

I created two clean git branches, and into each merged half of the code from the original branch that I was trying to merge. Each of these smaller branches was reviewed and merged (though I should have tried to break it up into three or four branches rather than just two). Unfortunately, when trying to merge these smaller branches back together, rather than seamlessly reassembling to get the initial branch as I'd intended, I was faced with numerous merge conflicts.

I learned the importance of considering the people reviewing my code when writing code. In future, I will do my best to ensure that all pull requests are small and to the point, containing only the code that they need to implement the feature I'm attempting. When I am blocked and decide to work on unrelated features in the code base, I will do this work on separate branches.

The following EA stage one competencies were developed during this event.

EA 2.2 Fluent application of engineering techniques, tools and resources.

EA 2.4 Application of systematic approaches to the conduct and management of engineering projects.

8 Conclusion

With the Mychro system, genomic data is now more accessible than ever. Insights from a patients genetic variations, particularly CNVs, can assist clinicians in treatment and diagnosis of genetic illnesses and particular cancers By using a to scale graphical representation of genetic data, the Mychro system can display and highlight the relationships between CNVs and genes discovered by past genetic research.

Through the use of an in memory R tree index, the Mychro system is able to respond to requests relating to genetic data in soft real time. Kubernetes platform and Prometheus service allow for a reliable and secure deployment of the Mychro system, as well as supporting in depth introspection of the running Mychro system. The use of GraphQL enabled high levels of abstraction internally within the Mychro system, and related tools such as the Hasura Engine and the Apollo Gateway allowed for automation of entire services within the system.

The agile methodology used to develop the Mychro system resulted in high developer productivity, and a low risk project. Flexibility in the goals and priorities allowed for fast response to user feedback. Early investigation and mitigation of risks allowed for all necessary changes to be made without significant loss of prior work. Integration with an established software development team afforded many opportunities for professional development during the project.

The Mychro system has room for improvements and future work. Such improvements could investigate analysis of SNVs, and methods of simplifying use such as the integration of a textual search feature.

Bibliography

Beckmann, N., Kriegel, H.-P., Schneider, R. & Seeger, B. (1990), 'The r*-tree: An efficient and robust access method for points and rectangles', *SIGMOD Rec.* **19**(2), 322–331.
URL: *http://doi.acm.org/10.1145/93605.98741*

Beyer, B., Jones, C., Petoff, J. & Murphy, N. (2016), *Site Reliability Engineering: How Google Runs Production Systems*, O'Reilly Media, Incorporated.
URL: *https://books.google.com.au/books?id=81UrjwEACAAJ*

Bluestone, J. A., Herold, K. & Eisenbarth, G. (2010), 'Genetics, pathogenesis and clinical interventions in type 1 diabetes', *Nature* **464**, 1293 EP –.

Brazil, B. (2016), 'Logs and metrics and graphs, oh my', *Grafana Labs Blog* .
URL: *https://grafana.com/blog/2016/01/05/logs-and-metrics-and-graphs-oh-my/*

Burk, N. (2019), 'Prisma 2 preview: Type-safe database access & declarative migrations', *Prisma Blog* .
URL: *https://www.prisma.io/blog/announcing-prisma-2-zq1s745db8i5/*

Carmichael, J. (2017), 'Occupational health and safety risk management', *UQ Policy and Procedures Library* **2.30.01**.
URL: *http://ppl.app.uq.edu.au/content/2.30.01-occupational-health-and-safety-risk-management*

Consortium, G. P., Auton, A. & Brooks, L. (2015), 'A global reference for human genetic variation', *Nature* **526**(7571), 68–74.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2009), *Introduction to Algorithms, Third Edition*, 3rd edn, The MIT Press.

Derstappen, T. (2018), How to build up-to-date (container) applications, *in* 'DevOpsCon'.
URL: *https://devopsconference.de/wp-content/uploads/2018/07/DOC_2018_Whitepaper.pdf*

Facebook, Inc & GraphQL Foundation (2018), 'Graphql'.
URL: *https://graphql.github.io/graphql-spec/June2018/*

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999), 'Rfc 2616, hypertext transfer protocol – http/1.1'.
URL: *http://www.rfc.net/rfc2616.html*

Flynn-Curran, R. et al. (2019), 'The package manager for kubernetes'.
URL: *https://helm.sh/*

Google Cloud (2019), 'Machine types', *Compute Engine Documentation* .
URL: *https://cloud.google.com/compute/docs/machine-types*

Guttman, A. (1984), *R-trees: a dynamic index structure for spatial searching*, Vol. 14, ACM.

Hasura, Inc (2016), 'What is hasura', *Hasura Blog* .
URL: *https://blog.hasura.io/what-is-hasura-ce3b5c6e80e8/*

Hasura, Inc (2018), 'The ultimate guide to schema stitching in graphql', *Hasura Blog* .
URL: *https://blog.hasura.io/the-ultimate-guide-to-schema-stitching-in-graphql-f3*

Heisler, B. (2018), Analysis process, *in* 'Criterion User Guide'.
URL: *https://bheisler.github.io/criterion.rs/book/analysis.html*

Hubbard, T., Barker, D., Birney, E., Cameron, G., Chen, Y., Clark, L., Cox, T., Cuff, J., Curwen, V., Down, T. et al. (2002), 'The ensembl genome database project', *Nucleic acids research* **30**(1), 38–41.

Illumina, Inc (2018), 'Trusight sequencing panels'.
URL: *https://sapac.illumina.com/products/trusight-panels.html*

International Human Genome Sequencing Consortium (2004), 'Finishing the euchromatic sequence of the human genome', *Nature* **431**, 931 EP.

Kanehisa, M. & Goto, S. (2000), 'KEGG: Kyoto Encyclopedia of Genes and Genomes', *Nucleic Acids Research* **28**(1), 27–30.
URL: *https://dx.doi.org/10.1093/nar/28.1.27*

Kim, K.-C. & Yun, S.-W. (2005), Mr-tree: A cache-conscious main memory spatial index structure for mobile gis, *in* 'Web and Wireless Geographical Information Systems: 4th International Workshop, W2GIS 2004, Goyang, Korea, November 26-27, 2004, Revised Selected Papers', Vol. 3428 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 167–180.

Landrum, M. J., Lee, J. M., Riley, G. R., Jang, W., Rubinstein, W. S., Church, D. M. & Maglott, D. R. (2013), 'Clinvar: public archive of relationships among sequence variation and human phenotype', *Nucleic acids research* **42**(D1), D980–D985.

Lee, J. W., Aminkeng, F., Bhavsar, A. P., Shaw, K., Carleton, B. C., Hayden, M. R. & Ross, C. J. D. (2014), 'The emerging era of pharmacogenomics: current successes, future potential, and challenges', *Clinical genetics* **86**(1), 21–28.
URL: *https://www.ncbi.nlm.nih.gov/pubmed/24684508*

Lejeune, J. & Turpin, R. (1961), 'Chromosomal aberrations in man', *American journal of human genetics* **13**(1 Pt 2), 175.

Li, H. (2013), 'Aligning sequence reads, clone sequences and assembly contigs with bwa-mem', *arXiv preprint arXiv:1303.3997* .

Nairn, B. (2018), 'Running your modern .net application on kubernetes'.
URL: *https://cloud.google.com/files/whitepaper-running-your-modern-dotnet-app-on*
pdf

Pop, M. & Salzberg, S. L. (2008), 'Bioinformatics challenges of new sequencing technology', *Trends in genetics : TIG* **24**(3), 142–149.

Prometheus Authors (2019), Metric types, *in* 'Prometheus Docs'.

Puliafito, A., Casalicchio, E. & Trivedi, K. S., eds (2019), *Container Orchestration: A Survey*, Springer International Publishing, Cham, pp. 221–235.
URL: *https://doi.org/10.1007/978-3-319-92378-9_14*

Rao, J. & Ross, K. (2000), 'Making b+- trees cache conscious in main memory', *ACM SIGMOD Record* **29**(2), 475–486.
URL: *http://search.proquest.com/docview/29399990/*

Reece, J. B., Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V. & Jackson, R. B. (2014), *Campbell Biology*, 10 edn, Pearson Learning Solutions.

Roussopoulos, N. & Leifker, D. (1985), 'Direct spatial search on pictorial databases using packed r-trees', *ACM SIGMOD Record* **14**(4), 17–31.

Sherry, S. T., Ward, M.-H., Kholodov, M., Baker, J., Phan, L., Smigielski, E. M. & Sirotkin, K. (2001), 'dbsnp: the ncbi database of genetic variation', *Nucleic acids research* **29**(1), 308–311.

Stadler, Z. K., Schrader, K. A., Vijai, J., Robson, M. E. & Offit, K. (2014), 'Cancer genomics and inherited risk', *Journal of Clinical Oncology* **32**(7), 687.

Strachan, T. & Read, A. (2011), 'Human molecular genetics'.

Sturm, R. A. (2009), 'Molecular genetics of human pigmentation diversity', *Human Molecular Genetics* **18**(R1), R9–R17.
URL: *https://doi.org/10.1093/hmg/ddp003*

Tassabehji, M., Metcalfe, K., Karmiloff-Smith, A., Carette, M. J., Grant, J., Dennis, N., Reardon, W., Splitt, M., Read, A. P. & Donnai, D. (1999), 'Williams syndrome: use of chromosomal microdeletions as a tool to dissect cognitive and physical phenotypes', *The American Journal of Human Genetics* **64**(1), 118–125.

The PostgreSQL Global Development Group (2018), Frontend/backend protocol, *in* 'PostgreSQL: Documentation'.

Wetterstrand, K. A. (2018), 'DNA sequencing costs: Data from the NHGRI genome sequencing program (GSP)'.
URL: *https://www.genome.gov/sequencingcostsdata/*

World Health Organization et al. (2002), *Genomics and World Health*, World Health Organization.

A Orderings on Ranges

Our goal is to prove that it is not possible to use an index in one dimension to solve range intersection queries. Given a set of ranges $R \subseteq \mathbb{N}^2$, a range query is defined by its input $r \in \mathbb{N}^2$ and its output $R' \subseteq R$ where all elements of R' intersect with the target range r. Add a figure here to explain in more detail, with an example.

A one dimensional index is defined by a total ordering on R, typically in the form of a binary tree. This index is notable for its ability to efficiently return subsets of R of the form $\{x \in R | x > a \wedge x < b\}$ given some values a, b. A solution to range intersection queries using a one dimensional index is defined as a combination of a total ordering ρ on R, and a mapping M from query ranges r to values a, b that can be used in filtering.

Consider, for our proof, a set R of four ranges, defined in eq. (A.1).

$$R = \{[0, 10], [8, 17], [12, 14], [15, 20]\} \tag{A.1}$$

For this set R, a series of range queries α, β, γ are defined in eq. (A.2). We prove that it is not possible to define a consistent total ordering on R that will answer solve these range queries through exhaustion, by analysing all possible total orderings.

$$\alpha = [5, 9] \tag{A.2}$$
$$\beta = [11, 13] \tag{A.3}$$
$$\gamma = [16, 21] \tag{A.4}$$

These range queries have been selected so that they each intersect with exactly two of the ranges from R. For any total ordering on R, we can assess whether that ordering can be used to answer a given range query by ensuring that the two target ranges for that query are adjacent to each other in the ordering. If the target ranges are adjacent, then it is possible to insert a and b into the ordering such that the output of the query will be the target ranges. If the target ranges are not adjacent, any selected values of a and b will also include one or more invalid ranges in the output set.

In table A.1, each permutation of R is analysed for each of the range queries, to determine whether that permutation is a valid total ordering to answer that range query. Through exhaustive analysis, it is shown that it is impossible to define a total ordering on R that will correctly answer all three range queries.

Table A.1: Exhaustive analysis of permutations of R

Permutation	α	β	γ
$\langle[0,10],[8,17],[12,14],[15,20]\rangle$	valid	valid	invalid
$\langle[0,10],[8,17],[15,20],[12,14]\rangle$	valid	invalid	valid
$\langle[0,10],[12,14],[8,17],[15,20]\rangle$	invalid	valid	valid
$\langle[0,10],[12,14],[15,20],[8,17]\rangle$	invalid	invalid	valid
$\langle[0,10],[15,20],[8,17],[12,14]\rangle$	invalid	valid	valid
$\langle[0,10],[15,20],[12,14],[8,17]\rangle$	invalid	valid	invalid
$\langle[8,17],[0,10],[12,14],[15,20]\rangle$	valid	invalid	invalid
$\langle[8,17],[0,10],[15,20],[12,14]\rangle$	valid	invalid	invalid
$\langle[8,17],[12,14],[0,10],[15,20]\rangle$	invalid	valid	invalid
$\langle[8,17],[12,14],[15,20],[0,10]\rangle$	invalid	valid	invalid
$\langle[8,17],[15,20],[0,10],[12,14]\rangle$	invalid	invalid	valid
$\langle[8,17],[15,20],[12,14],[0,10]\rangle$	invalid	invalid	valid
$\langle[12,14],[0,10],[8,17],[15,20]\rangle$	valid	invalid	valid
$\langle[12,14],[0,10],[15,20],[8,17]\rangle$	invalid	invalid	valid
$\langle[12,14],[8,17],[0,10],[15,20]\rangle$	valid	valid	invalid
$\langle[12,14],[8,17],[15,20],[0,10]\rangle$	invalid	valid	valid
$\langle[12,14],[15,20],[0,10],[8,17]\rangle$	valid	invalid	invalid
$\langle[12,14],[15,20],[8,17],[0,10]\rangle$	valid	invalid	valid
$\langle[15,20],[0,10],[8,17],[12,14]\rangle$	valid	valid	invalid
$\langle[15,20],[0,10],[12,14],[8,17]\rangle$	invalid	valid	invalid
$\langle[15,20],[8,17],[0,10],[12,14]\rangle$	valid	invalid	valid
$\langle[15,20],[8,17],[12,14],[0,10]\rangle$	invalid	valid	valid
$\langle[15,20],[12,14],[0,10],[8,17]\rangle$	valid	invalid	invalid
$\langle[15,20],[12,14],[8,17],[0,10]\rangle$	valid	valid	invalid

The proof for this small finite set can easily be expanded to the full set of possible ranges on $\mathbb{N}$. Consider any given ordering on the set of possible ranges on $\mathbb{N}$. Depending on the order that it defines on the elements of R, it can be classified as similar to one of the above 24 permutations. Depending on the permutation, it is possible to show that for at least one of the range queries α, β, and γ, at least one range from R will be incorrectly included in the response to the range query.

B Previous Prototype

In early 2018, a visualiser for the human genome was produced by Aginic. This visualiser was implemented as a custom visualisation extension to business analytics platform Qlik Sense. Due to the spatial nature of the data, the Qlik Sense was unable to perform certain queries and visualisations. The project suffered significant performance issues, as the application took between 8 and 30 seconds to load. Benchmarking and profiling showed that the performance issues were caused by a variety of issues, detailed below.

- Qlik Sense was not able to filter data with sufficient complexity, meaning that the application needed to download and process data points that would not fit on the screen to be rendered.
- The indexing model used by Qlik Sense was not able to provide speed increases for range based genomics data, such as gene lists.
- The data returned by the Qlik Sense API attached large amounts of unnecessary data, leading to increased latency fetching data and processing it.

Due to these performance issues, and the licensing costs of the Qlik Sense engine, the project was cancelled.

C Block Sizes

Modern operating systems provide utilities to report system information to users. These were used to analyse the size of CPU cache lines and file system block sizes on a series of modern computers. On macOS Mojave, listing C.1 was used to report cache line size in bytes, and listing C.2 was used to report file system block size.

Listing C.1: Cache line size

```
$ sysctl hw.cachelinesize
```

Listing C.2: File system block size

```
$ diskutil info / | grep 'Device Block Size'
```

These commands were run on a series of macOS devices, and the results are listed in table C.1.

Table C.1: Sizes of CPU cache lines and file system blocks

Computer	Cache line size (bytes)	File system block size (bytes)
MacBook Pro (Retina, 13-inch, Early 2015)	64	4096
MacBook Pro (13-inch, 2017)	64	4096
MacBook Air (13-inch, early 2015)	64	4096

D Index Performance Results

For each measurement, the benchmarked code was run multiple times in a tight loop. Table D.1 shows the aggregated results of each test, with outliers removed.

Table D.1: Results

Indexing scheme	Range (MB)	Number of tests (1000s)	Minimum time (μs)	Mean time (μs)	Maximum time (μs)
No index, no ordering	1-2	631	7.1185	7.1826	7.2619
	50-51	712	7.0724	7.1205	7.1798
	200-203	692	7.2467	7.4873	7.7657
No index, genes sorted by start	1-2	712	6.9486	6.9732	7.0026
	50-51	707	7.2907	7.6348	8.0905
	200-203	424	8.9662	9.8390	10.858
B Tree index, genes sorted by start	1-2	9 300	0.54090	0.54590	0.55228
	50-51	616	7.5757	7.6983	7.8306
	200-203	212	22.087	22.759	23.536
R Tree index	1-2	4 300	1.1775	1.1826	1.1876
	50-51	5 700	0.81601	0.83185	0.85093
	200-203	3 700	1.3067	1.3303	1.3570

E Interim System Benchmarks

An investigation was performed into the performance of the application servers under load half way through the project. A sample dataset (the global genome view) was queried from the servers, to measure the latency of the request. Multiple tests were performed, each with differing levels of concurrency in requests, to measure how latency was affected by load. To reduce variance, each test queried the server 10,000 times. To avoid measuring external network latency, all measurements were performed from a virtual CPU within the same cloud region as the application server. The latency of the application server is displayed in fig. E.1.

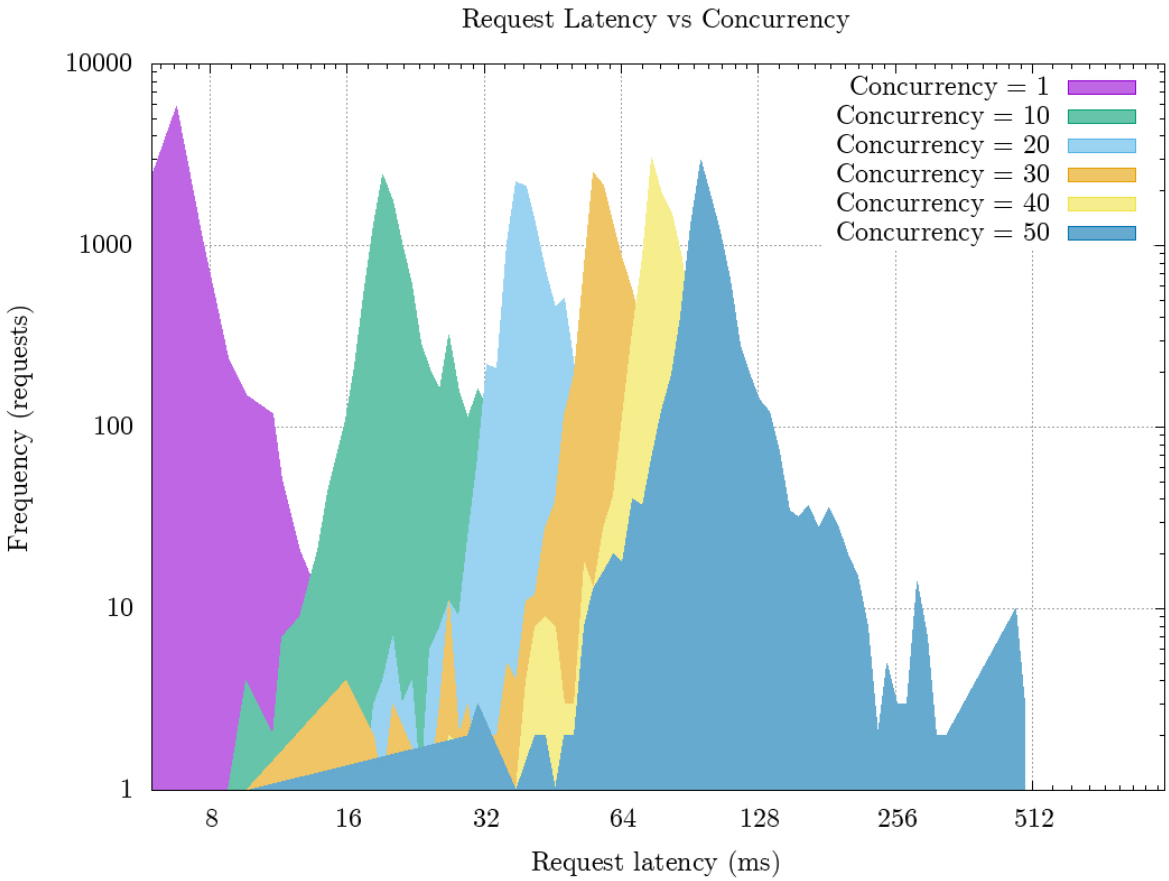

Figure E.1: Performance testing of application server under load

This performance data shows that larger numbers of concurrent connections result in higher latency for user requests. The sources of latency in these tests can be roughly divided into three categories, shown below.

Application latency This latency is the time from the application server receiving a request to the application server responding to that request.

Reverse proxy latency This latency is the time between the reverse proxy receiving and responding to a request that is not spent waiting for the application server.

Larger network latency This latency is the time when packets are in transit between the client and the reverse proxy, determined largely by factors such as the quality of the client's internet connection, and the distance between the client and the reverse proxy.

To determine the primary causes of request latency, monitoring of the servers during this testing was performed. The 90th percentile of request latency was measured by both the application server and the reverse proxy throughout the load testing, and the results are displayed in table E.1.

Table E.1: Server side monitoring during load testing

	90th Percentile latency (ms)	
Concurrent connections	Reverse proxy	Application server
1	5	2
10	21	2
20	34	2
30	48	2
40	52	2
50	100	2

From this data, it can be seen that the application server is not a latency bottleneck at scale. At this stage in the project, the performance of the application server itself is sufficient. As such, there is currently no need to implement more complicated optimisations to the R tree data structure or the rest of the application.