University of Dublin

TRINITY COLLEGE

Automating the parallel alignment of DNA using Kubernetes

Aideen Fay

B.A.I. Computer Engineering

Final Year Project April 2020

Supervisor: Jeremy Jones

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I, Aideen Fay, hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Aideen Fay _____ 25/04/2020 _____

Signed Date

Acknowledgements

Firstly, I would like to thank my supervisor Jeremy Jones who encouraged me to take on this project. Thank you for giving me the freedom to explore my ideas and for being a constant support throughout the academic year.

Thank you to my classmates and close friends who never failed to lighten my heart, clarify my doubts and inspire me to work harder.

Finally, I want to thank my family - Laurence, Janet, Ciara, Aoife, Clara and Laurence Jr. for supporting me from the beginning and for always bringing me home to myself. I want to thank my partner for his unwavering support and investment in my happiness and success. Thank you for being with me every single step of the way.

ABSTRACT

AUTOMATING THE PARALLEL ALIGNMENT OF DNA USING KUBERNETES

By Aideen Fay

Supervised by Jeremy Jones

Trinity College Dublin | School of Computer Science and Statistics

B.A.I. Computer Engineering

Read alignment is the first stage in the DNA analysis pipeline responsible for processing the ever-increasing amount of genomic data produced by next generation sequencing technologies. Currently available alignment tools are not capable of processing the available genomic data quickly and cheaply enough, some requiring days to complete a full alignment[1].

BWBBLE is a read alignment program that uses a collection of genomes to reduce the bias and improve accuracy when performing read alignment. The benefits BWBBLE offers are dwarfed by the fact that it is 100 times slower than other read alignment programs which use a single reference genome[2]. This is compounded by the fact that read alignment is already the bottleneck stage of DNA analysis.

To minimize the time taken to perform short-read alignments with BWBBLE, various parallelized execution models such as AWS-BWBBLE[3] and SparkBWBBLE[4] have been developed. While AWS-BWBBLE succeeded in demonstrating the potential for a linear speedup by distributing reads amongst several cloud based virtual machines, it introduced significant cost and infrastructure management overhead. As a result of this overhead, adoption has been limited and the practical speedups achieved remain out of reach of DNA researchers.

This project successfully demonstrated that a linear speedup in BWBBLE is possible by parallelizing the read alignment stage using Kubernetes jobs. We also demonstrated that it was possible to achieve this cheaply and with minimal operational overhead using a Kubernetes Controller. In doing so, we have made it possible for more researchers to easily leverage BWBBLE for their work.

---

[1] Arram, J., Kaplan, T., Luk, W. & Jiang, P., 2017. Leveraging FPGAs for Accelerating Short Read Alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatic,* May.14(3).

[2] Huang, L., Popic, V. & Batzoglou, S., 2013. Short read alignment with populations of genomes. *Bioinformatics,* July, 29(13), p. i361–i370.

[3] McGinley, K., 2019. *Parallel DNA Read Alignment Using the Amazon Cloud,* MSc Computer Science, Trinity College Dublin.

[4] Stratford, B., 2018. Cloud based high-speed parallel DNA read alignment, B.A.I. Computer Science, Trinity College Dublin.

Table of Contents

Glossary of Terms

Deoxyribonucleic Acid	**DNA**	*The genetic code needed to build and maintain and organism.*
Burrows-Wheeler Aligner	**BWA**	*A short-read alignment program which maps low-divergent sequences against a large reference genome.*
Amazon Web Services	**AWS**	*A cloud service provider that offers cloud computing services.*
Google Cloud Platform	**GCP**	*Google's cloud service offering that provides computing resources for web applications.*
Virtual Machine	**VM**	*A software program that imitates dedicated hardware on a host.*
Variant Call Format	**VCF**	A text file format used in sequence analysis which stores gene sequences.
Message Passing Interface	**MPI**	*A specification for message passing libraries.*
Elastic File Share	**EFS**	*A cloud-based file storage service offered by AWS.*
Azure Active Directory	**AAD**	*Microsoft Azure's identity and access management platform.*
Azure Kubernetes Service	**AKS**	*Microsoft Azure's managed Kubernetes Service.*
Central Processing Unit	**CPU**	*Primary computer component that processes instructions.*
Command Line Interface	**CLI**	*A command line program that accepts text to manage and view files.*
Random Access Memory	**RAM**	*A computer's short-term data store providing quick access to data.*
Operating System	**OS**	*Low-level software that controls and manages a computers core functionality.*
Application Programming Interface	**API**	*An interface that defines how software component should interact.*
Certification Authority	**CA**	*A CA manages, validates, issues and revokes certificates.*
Certificate Signing Request	**CSR**	*The process of requesting a certificate from a CA.*
Azure Container Instances	**ACI**	*Azure offering for rapidly creating and deploying containerized apps.*
etcd	**etcd**	*Kubernetes' distributed key-value store that holds all cluster data.*

Google Compute Engine	**GCE**	*Google's VM offering that provides scalable compute resources.*
Persistent Disk	**PD**	*The primary storage used by GCE virtual machines.*
Role Based Access Control	**RBAC**	*Method of access control based on user roles and permissions.*
Server Message Block	**SMB**	*A network protocol that allows nodes on a network shared access to files.*
Virtual Machine	**VM**	*A software program that imitates dedicated hardware on a host.*
Yet Another Markup Language	**YAML**	*A data serialization language which is a superset of the JSON format.*

List of Figures

1 Introduction

This project deals with the BWBBLE read alignment program. Read alignment is the process of locating the position of a "read" (a sequence of DNA) in a reference genome. This is done to discover the variation of the newly sequenced genome with respect to a previously sequenced reference genome. Unlike other read alignment programs such as SOAP2 (Li, et al., 2009), BWA (Li & Durbin, 2009) and Bowtie (Langmead, et al., 2009), BWBBLE uses a collection of two or more (up to millions of) genomes called a multi-reference genome. This eliminates the inherent bias to one specific genome and yields higher accuracies when aligning reads (Huang, et al., 2013).

The benefits BWBBLE offers is dwarfed by the fact that it is 100 times slower than other read alignment programs (Huang, et al., 2013). This is compounded by read alignment being the bottleneck stage of DNA analysis. Read alignment faces major computational challenges in processing the ever-increasing amount of genomic data made available by next generation high-throughput sequencing technologies. Currently available alignment tools aren't capable of processing the available genomic data in a cost and time effective manner, some requiring days to complete a full alignment (Arram, et al., 2017).

To minimize the time taken to perform short-read alignments, various read alignment programs have been deployed to cloud based platforms to exploit their abundant and highly scalable compute resources (Chen, et al., 2015), (Popic & Batzoglou, 2017). Using a massively parallel approach, the reads can be aligned by distributing them amongst cloud based VMs which can then process them concurrently. In this way, read alignment programs are no longer limited by the amount of hardware that a researcher can afford to purchase and operate themselves.

In an effort to increase the speed of BWBBLE, various parallelized execution models such as AWS-BWBBLE (McGinley, 2019) and SparkBWBBLE (Stratford, 2018) have been developed. These leverage the compute capacity of Amazon Web Services (AWS) and Google Cloud Platform (GCP) respectively. While AWS-BWBBLE succeeded in demonstrating the potential for a linear speedup by distributing reads amongst several cloud based virtual machines, it introduced significant cost and infrastructure management overhead. As a result of this overhead, adoption has been limited and the practical speedups achieved remain out of reach of DNA researchers.

This project uses Microsoft Azure and modern orchestration tools to build a scalable and reliable BWBBLE program that has the capability to process heavy workloads whilst abstracting the infrastructure management overhead away from the user.

1.1 Research Objectives

This project has three primary objectives. The first objective aims to reduce the time taken to run a read alignment using BWBBLE. The second objective is to dramatically reduce the time investment required by users to run and parallelize BWBBLE on the cloud. The final objective wants to make a cloud based BWBBLE as accessible to users as possible from a usability and affordability standpoint. This largely focuses on reducing the technical and infrastructure management overhead.

1. **Speed up BWBBLE**

   We want to achieve a linear speedup in the BWBBLE program. This fundamentally involves deploying BWBBLE to the cloud and highly parallelizing it by distributing the reads amongst Kubernetes Pods for concurrent processing. This is done with the intention of dramatically reducing the time taken to complete a read alignment with BWBBLE - making it practical for real-world use.

2. **Reduce the end-end time for a geneticist to run BWBBLE on the cloud**

   We want to reduce the end-to-end time taken for anyone to complete a read alignment with BWBBLE. This builds upon our first objective and expands it, focusing on reducing the total time taken for an end user to run BWBBLE in the cloud. The goal is to provide a simplified API that enables geneticists to perform alignments without needing to manually execute steps in the alignment process or manage failures. This enables geneticists to focus on the read alignment results rather than operating and managing the program itself.

3. **Make BWBBLE more accessible for use at scale**

   Current solutions to parallelize BWBBLE require users to modify the source code to add this functionality, as well as build and deploy this to cloud infrastructure which they must configure and maintain. This is a time-consuming process which assumes the user has significant technical skills. With this project, we want to reduce the technical management overhead demanded by prior solutions so that any researcher can gain the benefit of this read alignment program. To further increase accessibility to a cloud based BWBBLE we want to design the project to run as cheaply as possible; allowing any scientist without their own dedicated hardware to use the program on a limited budget.

1.2 Project Requirements

The project requirements below are described in RFC2119[5] form.

- The solution MUST leverage BWBBLE to perform DNA alignment.
- The solution MUST use Microsoft Azure as the public cloud provider.
- The solution MUST take advantage of multiple machines to parallelize its work.
- The solution MUST provide a tangible reduction in total runtime for alignment jobs over prior solutions including AWS-BWBBLE and SparkBWBBLE.
- The solution SHOULD reduce the technical and infrastructure management overhead for running a cloud based BWBBLE alignment job over prior solutions including AWS-BWBBLE and Spark-BWBBLE.
- The solution SHOULD address the privacy and security considerations of the proposed design architecture.
- The solution MAY use Kubernetes to provide job orchestration and scheduling functionality.
- The solution MAY provide a RESTful API through which alignment jobs are submitted.
- The solution MAY take advantage of caching to reduce the time taken to execute jobs for which previous steps have already been completed (e.g. caching of previously indexed reference genomes for future use).

[5] The RFC2119 specification for outlining requirements can be found at https://tools.ietf.org/html/rfc2119.

1.3 Report Overview

The report is divided into several sections, providing background information, a design overview, details of the implementation and finishing with benchmarking and conclusions.

Chapter 2.1 introduces various DNA and DNA sequencing concepts that are necessary to understand the process of read alignment and the significance of the challenges it faces.

Chapter 2.2 outlines the BWBBLE program and its workload stages. This chapter also explores the prior works that have focused on reducing the time taken to perform a read alignment with BWBBLE.

Chapter 2.3 introduces cloud computing and the challenges of managing and running workloads at today's scale. It outlines the need for containerization and the modern orchestration tools like Kubernetes that manage these containerized workloads.

Chapter 3 gives an overview of the project's design architecture and why certain design decisions were made. It provides a high-level overview of the various stages in the project's implementation.

Chapter 4 describes the different stages involved in the project's implementation, from prototyping to final deployment. This chapter covers the technical aspects of how the project functions and provides insight into the code, for anyone wanting to implement a similar solution.

Chapter 5 outlines how the benchmarking was conducted and the results yielded by the benchmarking.

Chapter 6 contains the conclusions drawn from this project, as well as a reflection on the results achieved and areas in which this work may be expanded in future.

2 Background

2.1 DNA and Sequencing

Although this project focuses on how to build a reliable, cost effective and accessible system for running highly parallelizable scientific workloads, the subject of genetics is at the centre of BWBBLE. This section gives a brief introduction to various concepts related to BWBBLE and read alignment in general. This is done with the intention of giving the reader an understanding of the problems facing read alignment programs and familiarity with terms used later in this report.

2.1.1 DNA

Deoxyribonucleic acid or DNA is a molecule of two DNA strands coiled around each other to form a double helix structure. These strands of DNA are made up of nucleotides. Each nucleotide is composed of a phosphate group, a sugar group and one of four nitrogen bases: adenine (A), guanine (G), cytosine (C) and thymine (T). The order of these bases form genes and these genes hold the instructions for cells regarding the development, functioning and reproduction of an organism or virus. Each base is chemically bonded with another base to form a unit called a base pair. T always pairs with A and G is always the complement pair for C.

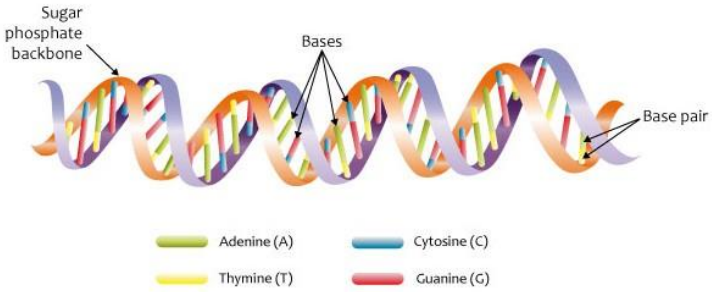

Figure 1 Simple structure of DNA (Genomics Education Programme)

2.1.2 DNA Sequencing

DNA sequencing is the process of determining the order of the four different nucleotide bases that appear in a DNA molecule. When a new genome is sequenced, the genome's DNA is fragmented into many millions of pieces or sequences of DNA. These sequences are called "short reads" - randomly arranged overlapping segments of the full DNA sequence. Short reads usually consist of 25-100 base pairs (Trapnell & Salzberg, 2009).

Before this information can be used, it must be ordered correctly – a process known as read alignment. These sequences determine the genetic information carried by a DNA segment and yield vast

information about the role of inheritance and environment in susceptibility to disease. These sequences can also highlight changes in genes that might cause disease.

Sequencing a DNA molecule results in a series of DNA sequences being recorded. These sequences determine the genetic information carried by a DNA segment and yield vast information about the role of inheritance and environment in susceptibility to disease. These sequences can also highlight changes in genes that might cause disease.

Due to the way in which DNA sequencing is performed, the result is a series of "short reads" – randomly arranged overlapping segments of the full DNA sequence. Before this information can be used, it must be ordered correctly – a process known as read alignment.

2.1.2.1 Reference genomes

A reference genome is the standard sequence of genes for an organism. It represents the full set of a species' genes and is used by read alignment tools to ensure that newly sequenced genomic data is assembled in the correct order.

2.1.3 Read Alignment

A read is a sequence of nucleotides obtained from an individual sequencing experiment. Short read alignment involves mapping a short "read" against a reference genome to locate the position that the read appears in the reference genome. A common analogy used is to imagine the short reads as the small pieces of a jigsaw puzzle and the reference genome as the completed jigsaw picture. The reads generated by sequencing techniques rely on being read aligned prior to being used for scientific work.

Read alignment the first step during genomic data analysis and can also be the bottleneck stage in DNA analysis programs. Next-generation sequencing (NGS) techniques have enabled the sequencing of thousands of human and other species' genomes (Stein, 2010). This has introduced an array of challenges for geneticists using the vast amount of genomic data produced (Durbin, 2009). One such challenge is that current read alignment programs struggle to efficiently map the short "reads" of DNA sequences produced by these high-throughput NGS technologies (Huang, et al., 2013). Despite current read alignment programs featuring highly optimised algorithms, they can take many hours or even days to align the short-read data (Arram, et al., 2017).

Along with being the bottleneck stage of DNA analysis at a time where a vast number of reads need to be processed, current read alignment techniques introduce various biases and yield lower accuracy as a consequence of using a single reference genome. These are some of the factors that led to the creation of the read alignment program that is at the core of this project; BWBBLE.

2.1.4 Burrows-Wheeler Transform

Mapping short reads against massive amounts of data in the form of reference genome is exhaustive and requires compression to make this difficult task practically feasible. The Burrows-Wheeler transform is essentially a compression tool that uses a data transformation algorithm to prepare data in such a way that it is more compressible. It acknowledges that it is easier to compress a string with runs of repeated characters and rearranges the string into sequences of similar characters. In BWBBLE and other BWT based read alignment programs, the BWT creates an indexed data structure of the reference genome to facilitate a linear-time alignment. The runtime complexity for alignment programs that do not make use of the BWT is $O(N_G)$, where N_G is the number of base pairs in the refence. In comparison, the runtime for alignment programs use the BWT is $O(N_R)$, where N_R is the number of base pairs in the read.

2.1.5 BWBBLE

BWBBLE is a BWT-based read alignment program developed by bioinformatic researchers at Harvard University that maps reads against a multi-reference genome. Other efficient short-read alignment programs which also use the BWT such as BWA, SOAP2 and BOWTIE already exist. However, these read alignment programs use a single reference genome for read alignment and therefore produce results that are inhernetly biased towards the specific genome used.

Biased alignment results can impede efforts to gain a comprehensive understanding of numerous domains including the cancer genome (Roach, et al., 2010), Mendelian disorders (Buckingham, et al., 2009) and evolutionary history (Kumar, et al., 2004). Alignment methods that use a single reference genome are also unsuitable for sequencing organisms that have large genomic variation among individuals and high polymorphism rates (Huang, et al., 2013).

The limitations of using a single reference genome become even more pronounced in cases where high quality mappings are required, as is the case in mouse strains (Keane, et al., 2011). Although the human genome has a lower polymorphism rate, reference bias remains a significant problem and can distort the conclusion of studies involving large numbers of individuals (Huang, et al., 2013). In such a study, the bias form individual alignment results can add up to distort the study's conclusion.

BWBBLE addresses the loss in accuracy and bias associated with using a single reference genome by instead aligning reads against a multi-reference genome. Due to the increased availability of reference genomes and genome variations, it is now more feasible to construct a multi-reference genome. Genome Mapper is another read alignment tool which uses a multi-reference genome created by integrating related genomes into a single graph structure (Schneeberger:, et al., 2009). However,

Genome Mapper has a high memory consumption and therefore isn't suitable or efficient for running human genome workloads due to the size of the genome (Li & Homer, 2010).

Due to the large amount of data BWBBLE processes when mapping reads against the multi-reference genome, it is up to 100 times slower than read alignment programs which use a single reference genome (Huang, Popic, & Batzoglou, 2013). However, BWBBLE is more efficient when we consider that for another single-reference genome based read alignment program to achieve the same accuracy as BWBBLE, it must process each reference genome in the collection individually.

2.2 BWBBLE

The BWBBLE program is split into two components called `mg-ref` and `mg-aligner`. Each component has a distinct role in performing a read alignment with BWBBLE and contains its own sub-components to do so.

The full alignment process is shown in Figure 2, with `mg-ref` stages shown in light blue and `mg-aligner` stages shown in red. The outputs of prior stages are passed to the next in the form of transient files.

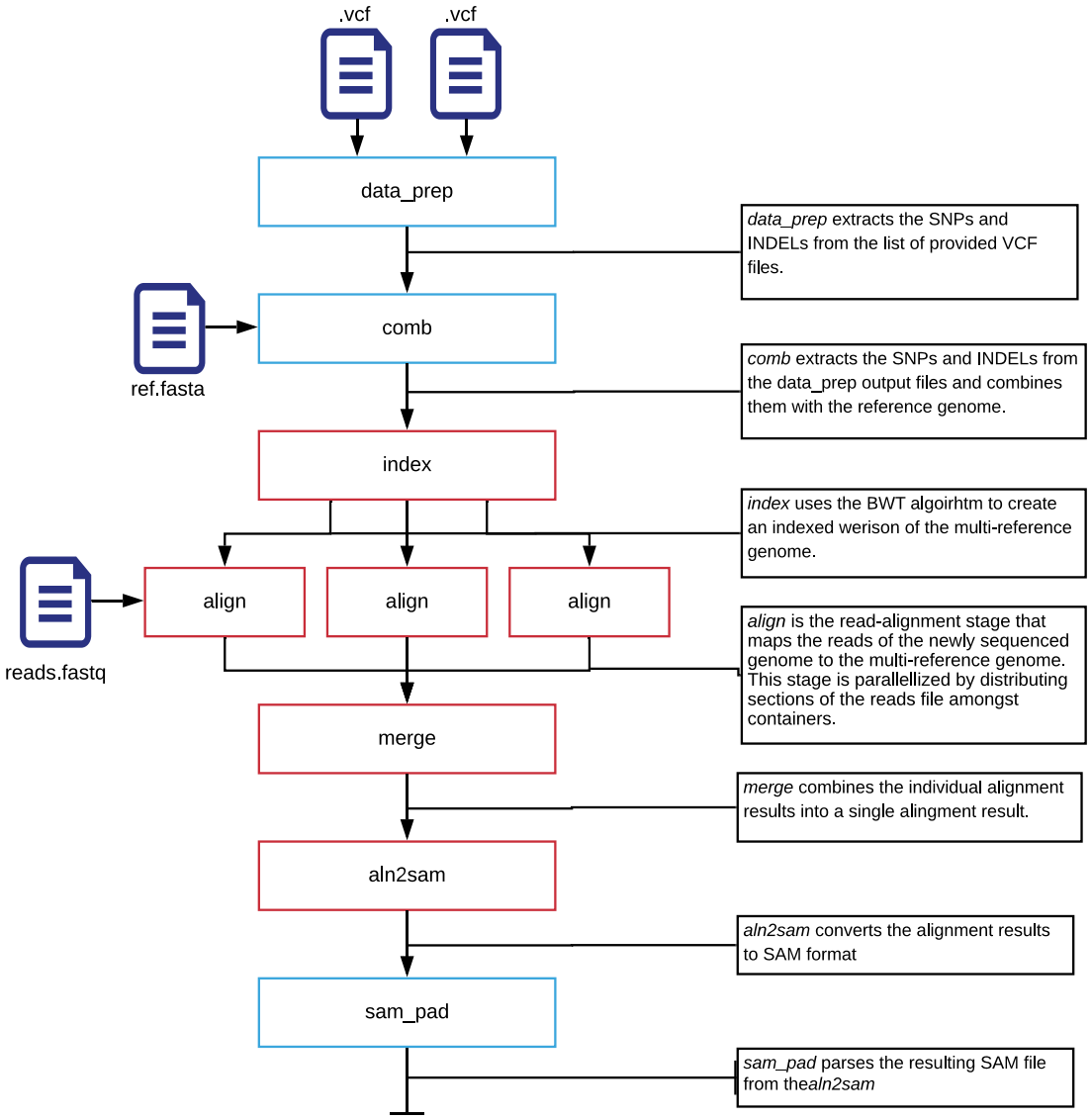

Figure 2 BWBBLE workload stages

2.2.1 mg-ref

This tool is responsible for preparing the multi-reference genome's FASTA file for use by the mg-aligner tool. It is also responsible for appending information from the multi-reference genome to the output generated by the aligner. When compiled, three separate binaries will be created: data_prep, comb and sam_pad.

2.2.1.1 data_prep

data_prep is the first stage of mg-ref. It extracts the SNPs and INDELs from the list of provided VCF files. A VCF file can include multiple variants from one or more chromones. This stage writes the SNPs into mg-ref-output/SNP.extract.chrxx.data and the INDELs into mg-ref-output/INDEL.extract.chrxx.data which will then be passed as inputs into the comb phase.

2.2.1.2 comb

The comb (or combination) stage reads the SNPs and INDELs from the extracted chromosome data in the mg-ref-output/SNP.extract.chrxx.data and mg-ref-output/INDEL.extract.chrxx.data files. This is then combined with the provided ref.fasta reference genome. The files outputted from this stage are ref_w_snp.fasta, ref_w_snp_and_bubble.fasta and bubble.data which combine the reference genome with the SNPs and INDELs.

2.2.1.3 sam_pad

This is the final stage of the BWBBLE program and it processes the SAM file which is generated by the aln2sam phase of mg-aligner. This command computes the positions of reads aligned to bubble branches in the reference genome and appends this information to the end of the corresponding lines in the SAM file (Huang, 2015).

2.2.2 mg-aligner

This tool is responsible for performing read alignment against the multi-reference genome. It accomplishes this by providing commands to index the multi-reference genome (using BWT) and align a series of reads against it. The functions themselves are index, align and aln2sam.

2.2.2.1 index

This stage uses the BWT algorithm to create an indexed data structure for the multi-reference genome's FASTA file. The alignment stage will use this to align the reads to the multi-reference genome.

2.2.2.2 align

The align stage maps a reads.fastq file, containing newly sequenced genomic data, against the multi-reference genome. This stage can be easily parallelized and is the primary target of this project. In this

project, we parallelize this stage by running multiple program instances within Azure Container Instances. By varying the number of ACIs used, we can adjust the amount of compute capacity available to BWBBLE as well as the cost of operation. BWBBLE also has a built-in multithreading option that splits the reads from the FASTQ reads file among parallel threads, enabling it to use multiple CPU cores within a single instance. This feature was not leveraged in this project as with was shown to perform sub-optimally in AWS-BWBBLE. Furthermore, the original BWBBLE journal article (Huang, et al., 2013) does not outline the implementation details or results of this multithreaded option.

To simplify the parallelization of this stage, the code for this portion of BWBBLE was modified to enable us to control which portion of the `reads.fastq` file each program instance would process. This was done by providing a "skip count" and a "process count" to each program instance.

Due to the way that the `output.aln` file is generated, it is possible to merge parallelized alignment outputs by concatenating the files. To achieve this, we used the Linux `cat` command.

2.2.2.3 aln2sam

This is the final stage of `mg-aligner` that converts the alignment results to SAM format for single-end mapping[6]. This SAM file may then have information from the multi-reference genome appended to it using the `sam_pad` tool provided by `mg-ref`.

2.2.3 Prior Works

To minimize the time taken to perform short-read alignments with BWBBLE and other read aligners, various parallelized execution models such as SparkBWA (Abuín, et al., 2016), SparkBWBBLE and AWS-BWBBLE have been developed.

2.2.3.1 SparkBWA

The Burrows-Wheeler aligner (BWA) is another read alignment tool based on the BWT. Unlike BWBBLE, it completes read alignments against a single reference genome. SparkBWA integrates BWA with Apache Spark, a big data processing framework, to run the program in parallel on public cloud computing clusters. In doing so, they hoped to improve the performance and scalability of BWA by taking advantage of parallel computing architectures and approaches such as Hadoop, `pthreads` and MPI. SparkBWA did not succeed in achieving a linear speed up in BWA, however it did achieve a slight performance improvement over its predecessor BigBWA (Abuín, et al., 2015) which integrated BWA with Hadoop rather than Apache Spark.

---

[6] Single end mapping means the sequencer reads the sequence of base-pairs from only one end to another without starting another read from the opposite end and back to the start like in paired-end reading.

2.2.3.2 SparkBWBBLE

SparkBWBBLE adapts the work of SparkBWA to run BWBBLE as the read alignment program instead of BWA. As part of this project, BWBBLE was deployed to run in parallel on a Google Dataproc[7] cluster of virtual machines with Spark pre-installed. The project results show that it did not achieve a linear speedup in the runtime of the BWBBLE program and concluded that Spark might not be the best solution to exploit BWBBLE's high parallelizability.

2.2.3.3 AWS-BWBBLE

AWS-BWBBLE builds on the work of Spark-BWBBLE to try and achieve a linear speedup in the BWBBLE program using Amazon Web Services (AWS) as a public cloud provider. It parallelizes the BWBBLE code, following a similar approach used in SparkBWBBLE, by distributing the reads amongst several cloud based virtual machines. While AWS-BWBBLE succeeds in demonstrating the potential for a linear speedup, it introduces significant cost and infrastructure management overheads.

The project assumes that the user is competent with using the AWS command line, Linux commands to orchestrate the splitting of the reads file and mounting the Amazon Elastic File Share to the virtual machines. As a result of this technical and time overhead, adoption has been limited and the practical speedups achieved remain out of reach of DNA researchers. The use of Docker to run the AWS-BWBBLE in containers is explored and concludes that the containerization doesn't introduce any benefits to speed up the alignment phase and complicates the access and automation (McGinley, 2019).

This project builds on the concepts AWS-BWBBLE explored. This project aims to leverage Kubernetes to enable an even higher degree of parallelism than was achieved with AWS-BWBBLE and reduce the infrastructure management and technical overhead it introduced for the user.

[7] Google Dataproc is a service offered by the Google Cloud Platform which manages Hadoop and Spark services.

2.3 Modern Job Orchestration

This section outlines the concepts and technologies that form the basis of the solution presented by this project. The state of the art in cloud computing is introduced including containerized workloads, Kubernetes, managed Kubernetes services and Kubernetes Custom Resources.

2.3.1 Cloud computing

Cloud computing is the delivery of computing resources such as servers, storage, networking, data analytics and software over the internet or "cloud". Cloud computing enables businesses to rent the scalable, reliable and highly available cloud services offered by cloud providers. Companies no longer need to pay large upfront costs and can focus on their product rather than managing the complexities of hardware and datacentres. Several cloud service providers exist that offer these services such as Amazon Web Services, Microsoft Azure and Google Cloud Platform.

2.3.2 Containers

As cloud providers began to offer elastic and on demand resource sharing, virtual machines (VMs) served as the building blocks of our cloud infrastructure enabling higher resource utilization while preserving the performance isolation among different computing instances (Zhang, et al., 2018). However, at scale it becomes difficult to manage, update and patch VMs. It is difficult to transfer resources from one virtual machine to another, despite much research into this problem of efficient resource management (Xiao, et al., 2013) (Peng, et al., 2012), leading to reduced system resilience. VMs can also struggle to handle rapidly shifting demand even when there are resources available (Zhang, et al., 2016). This is because they cannot scale to use these resources at the required rate.

Containers are like VMs but enable more efficient deployments that can operate at scale. Unlike VMs which use hardware-level virtualization, containers leverage OS-level virtualization (see Figure 3), with multiple containers running atop the OS kernel directly, making them more lightweight. This allows containers to use fewer resources and achieve higher CPU and memory utilization than a VM when running the same workload. They show better scalability than VMs and operations teams no longer need to manage a fleet of virtual machines with different operating systems and architectures. Using containers also enables the use of container orchestrators which provide an abstraction over the fleet, allowing users to treat entire datacentres as if they were a single computer.

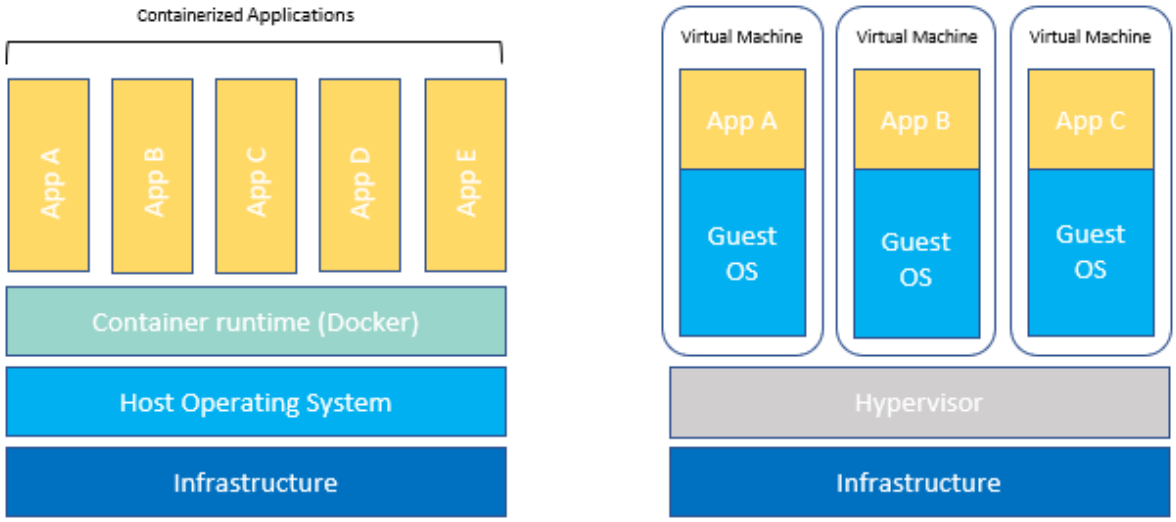

Figure 3 Operating System (OS) virtualization compared to hardware virtualization

Containers can be thought of an atomic unit of deployment and reuse that package up the software with all its dependencies, enabling the same application to run reliably, consistently and quickly in different environments. Developers do not have to maintain different versions of the software for different language versions, Linux distributions and libraries as the container only depends on the OS kernel. This makes containers attractive for use in scientific workloads as it can be challenging to achieve consistent results when reproducing scientific research in different cloud environments due to the dependencies included in scientific workloads and rapidly changing computer environments (Boettiger, 2014).

2.3.3 Kubernetes

Kubernetes is an orchestration platform for running containerized applications. Kubernetes is the by-product of lessons learned from Google's internal workload orchestration platform called Borg[8] which was built to admit, schedule, start, restart and monitor the full range of applications that Google runs (Pedrosa, et al., n.d.). Kubernetes is built on the principles of availability, security and portability and allows us to treat, interact with and reason about a cluster of machines as a single system. Kubernetes containers are not tied to individual machines but are abstracted across a cluster. Additionally, by building features like load-balancing and autoscaling into Kubernetes itself, we are no longer tied to a specific cloud-provider.

[8] Borg was Google's internal cluster management system that managed the starting, scheduling, restarting and monitoring of all Google's applications. More information can be found at https://pdos.csail.mit.edu/6.824/papers/borg.pdf

One of the attractions of using Kubernetes is that it is vendor agnostic. Once you have defined the resources you want to use, you can run and deploy your Kubernetes cluster on your local machine or a cloud platform without any tangible differences. This affords the user to have more freedom to address infrastructure cost and security concerns by leveraging a range of different offerings.

2.3.3.1 Kubernetes Architecture

A running Kubernetes deployment is called a cluster. The cluster can be visualized as two parts: the control plane and the worker nodes or compute machines. These worker nodes host pods which are responsible for running containerized workloads. By using a container as a unit of work, we can avoid the need for Kubernetes to understand the details of how our application should run, or the environment it should be run in.

To manage resources on the Kubernetes cluster, we interact with the Kubernetes API through the `kubectl` command-line client. The Kubernetes API allows us to create, monitor and delete Kubernetes resources.

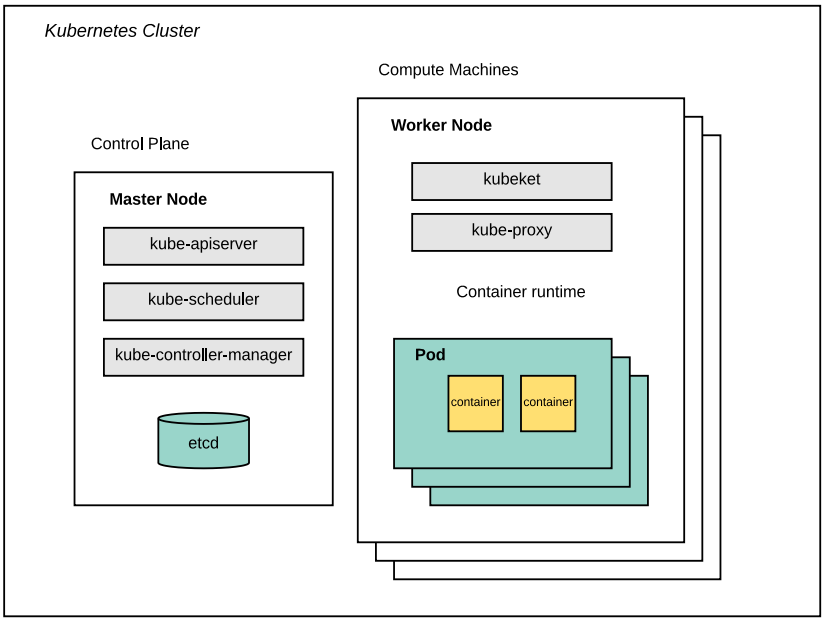

Figure 4 Kubernetes Architecture

2.3.3.1.1 Kubernetes Control Plane

The control plane is at the core of the Kubernetes cluster and is made of the `kube-apiserver`, the `kube-scheduler`, `etcd` and the `kube-controller-manager`. These components come together to form the master node.

To ensure that the Kubernetes control plane remains capable of operating during maintenance and outages, we run multiple master nodes. These master nodes perform leader election through `etcd`, a

fault-tolerant key-value store. `etcd` is also responsible for storing the configuration data and state information about the cluster such as what jobs are running, what nodes exist and the resources available on the cluster. It is the ultimate source of truth about the cluster.

Kubernetes is managed through an HTTP API which is provided by the `kube-apiserver`. The API handles both internal and external requests, allowing both users and controllers to interact with the cluster to create resources, run pods and check the status of running pods.

The `kube-scheduler` decides where to run the pods by considering information about the worker nodes in the cluster and selecting one based on the resource requirements of the pod and the health of the worker node.

Controllers in Kubernetes allow us to create higher-level descriptions of our work which are then executed in the form of Pods. There are several controllers in Kubernetes including those responsible for scaling pods or responding to their failures. The `kube-controller-manager` is responsible for running these controllers.

Containers are created by controllers through Kubernetes Pods. The most common controllers are `Deployments`, `StatefulSets` and `Jobs`. Jobs are used for run-to-completion applications and we build upon it in this project to execute DNA alignment workflows.

2.3.3.1.2 Worker Node

A worker node is the smallest unit of computing hardware in Kubernetes. It represents a single machine in the cluster and will be a physical machine in a datacentre or in this project's case, a virtual machine hosted by Azure. Thinking of the machine as a node allows us to abstract away the details of the underlying hardware and focus on application optimization and performance in terms of RAM and CPU allocation. A cluster must have at least one worker node if it is to execute user workloads.

When a workload is sent to the Kubernetes API, the scheduler will allocate the workload to a suitable node. Kubernetes automatically detects and reschedules workloads in the face of worker node failures. It should not matter to the user which individual machine is running the code. Pods are scheduled to run on the worker nodes and containers on these pods are what run the code.

2.3.3.1.3 Virtual Node

A virtual node is a service which appears to Kubernetes as if it were a Virtual Machine, but in fact represents an external service. They are particularly well suited to workloads which require rapid scaling of the cluster as they remove the need to provision and manage additional worker nodes directly.

In Azure Kubernetes Service, there is a virtual node which communicates with the Azure Container Instances service to run Kubernetes pods. These pods can be scheduled immediately and are charged per second of execution. This is in comparison to worker which are charged based on the time that the virtual machine is running for, regardless of whether it is running a pod. This reduces the cost of running a cluster with the ability to run dozens of concurrent workloads.

2.3.3.2 Node internals

Each node on the cluster has a `kubelet`, `kube-proxy` and a container runtime. These components are responsible for ensuring that the node performs the work assigned to it by the Kubernetes control plane.

`kubelet` is the most important node component and is responsible for communicating with the Kubernetes control plane to identify new work and report the status of workloads running on the node. When new work becomes available for the node, `kubelet` will communicate with the container runtime to start the relevant containers.

The container runtime is responsible for launching and managing the state of containers by communicating with the operating system kernel. There are several popular container runtimes that Kubernetes supports, including Docker and `containerd`.

The `kube-proxy` is responsible for coordinating cluster-wide networking, including the ability to expose network ports from within containers on the node.

2.3.3.3 Kubernetes Objects

Kubernetes objects are persistent entities in the Kubernetes cluster which represent the desired state of the cluster and its workloads. The Kubernetes objects this project uses are `Pods`, `Jobs`, `ConfigMaps` and our custom `AlignJob` resource type.

The concept of "maintaining desired state" is at the core of Kubernetes' declarative approach to workload execution. The objects describing this desired state are often defined in a YAML file, which may hold one or more Kubernetes objects. A YAML file containing Kubernetes objects may be loaded into the cluster by running the `kubectl apply -f my_k8s_object.yaml` command.

2.3.3.3.1 Pods

A `Pod` represents a single application instance within Kubernetes. Kubernetes does not manage containers directly but wraps them in a high-level structure called a `Pod`. Each `Pod` contains one or more containers and specifications that determine how the containers should run. If multiple containers run on the same `Pod`, they share the same resources and local network. `Pods` can be connected to persistent storage to maintain application state, if required.

2.3.3.3.2 Kubernetes Jobs

A `Job` describes a run-to-completion workload within Kubernetes. `Jobs` are often used to orchestrate batch operations and Kubernetes will execute one or more `Pods` to ensure that the `Job` completes successfully. This ensures that `Jobs` can recover automatically from application crashes, node failures or transient network issues.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: echo-Job
spec:
  template:
    spec:
      containers:
      - name: echo-job
        image: busybox
        command: ["echo", "Running a Kubernetes Job"]
      restartPolicy: Never
  backoffLimit: 4
```

Figure 5 YAML file creating a Kubernetes Job

Figure 5 shows a YAML file for creating a Kubernetes Job. The required fields in every Kubernetes YAML file are `apiVersion`, `kind`, `metadata` and `spec`. The `apiVersion` specifies which Kubernetes API group and version is used to create the object (`batch/v1` in this case). The `kind` field indicates which type of resource provided by the `apiVersion` we wish to create (in this case a `Job`). The `metadata` includes data used to uniquely identify the object in a Kubernetes cluster and enable easy categorization of related objects.

The object `spec` field provides the runtime specification which will be used to construct the `Pod`(s) that perform work within the scope of this `Job`. In this case the `Job` will run a single container using the `busybox` container image to print out "`Running a Kubernetes Job`".

The `restartPolicy` field specifies the conditions under which the container will restart. The `restartPolicy` is set to `Always` by default but can also be set to only restart `onFailure` or `Never`. When running as part of a `Job`, the restart policy should always be set to `Never`. The `backoffLimit` field specifies the number of times a container will retry a `Job` before it is considered failed.

2.3.3.3.3 Kubernetes ConfigMap

A `ConfigMap` is a way of storing configuration information within Kubernetes. It is essentially a dictionary of string key-value pairs which can hold any user-defined data. This configuration data may then be accessed by the `Pod` in which your container is running. The means by which the container accesses the `ConfigMap` values can vary depending on use case, with Kubernetes supporting environment variables and virtual files and two common means of access.

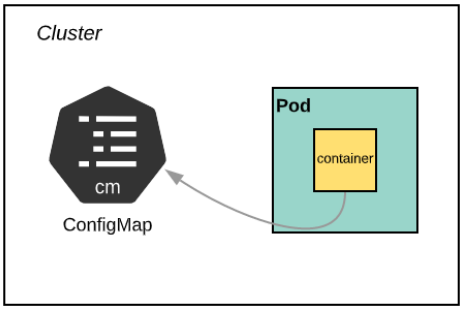

Figure 6 Container referencing Config Map configuration settings

2.3.3.3.4 Kubernetes Custom Resource

In addition to the core Kubernetes APIs, it is possible to define your own API groups and versions, with associated resource types. This enables you to extend the Kubernetes APIs with your own custom objects that will be stored for you by Kubernetes.

To define these custom resources, you create a Custom Resource Definition (CRD) object in Kubernetes. This CRD informs Kubernetes that your resource type exists, as well as providing information on its data schema.

Controllers may then be used to write logic that interacts with these custom resources, enabling you to build high-level logic on top of Kubernetes.

2.3.3.3.5 Kubernetes Secret

A Kubernetes `Secret` is a Kubernetes object that stores sensitive information such a as a password, access key or token. It is conceptually the same as a `ConfigMap` and can also be accessed by `Pods`. Unlike a `ConfigMap`, the values within a `Secret` are Base64 encoded to make supporting binary secrets (like cryptographic keys) easier.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Figure 7 A sample Kubernetes Secret created in YAML

2.3.3.4 Managed Kubernetes Services

Due to the complexity of setting up a Kubernetes cluster, there are several commercial Kubernetes offerings. These offerings provide a managed Kubernetes Control Plane and workers, with the service provider ensuring that components are updated regularly and configured correctly.

This project uses Azure Kubernetes Service, one such offering, however there are also offerings from Google, AWS and DigitalOcean amongst others.

3 Design

3.1 Application

To simplify the execution of BWBBLE in a cloud environment, we will containerize the various BWBBLE applications using Docker. Containers show better scalability than virtual machines and are more convenient to manage in deployment and bootup stages (Zhang, et al., 2018).

Packaging BWBBLE into a container image with its runtime dependencies, like OpenMP and `glibc`, ensures that it always executes in a consistent environment, ensuring that results are reproducible. Doing so also enables us to use standard industry orchestration frameworks like Kubernetes. This enables us to treat a fleet of virtual machines as if they were a single large computer, making it easier to manage highly parallelized workloads.

This means we no longer need to be concerned about the underlying hardware running the program but can focus on the performance of the software. The orchestration framework will handle to scheduling, running and failure recovery for the workload.

We also need to make changes to BWBBLE to enable it to process a subset of the `reads.fastq` file. Doing so enables us to run multiple BWBBLE aligner instances against the same file in parallel, distributing the work across a cluster of machines.

3.2 Orchestration

We will use Kubernetes to orchestrate the execution of the various containerized BWBBLE stages. This will be accomplished by creating Kubernetes `Jobs` for each of the stages. Kubernetes enables us to scale and parallelize the execution of these programs, optimally schedule them on appropriate nodes and handle transient failures automatically. By leveraging Kubernetes, we avoid the operational complexity problems inherent in virtual machines.

Running BWBBLE on Kubernetes abstracts the minutiae of running the BWBBLE application and managing VMs away from the user which makes running a BWBBLE read alignment much more accessible – both in terms of cost and technical skill. Kubernetes is well documented and has a strong development community which offers support to developers and users.

3.2.1 Management API

To simplify the user experience of submitting a reads file for alignment and abstract away the setting up of servers, mounting file systems, managing the number of containers that will run the workload and splitting the reads file amongst them - there needs to be an API. The Kubernetes API is a natural fit for this project's use case as it enables the definition of Custom Resource Types. This allows us to

streamline the execution of the BWBBLE program and remove the need for the user to manage the workflow or underlying VMs.

The BWBBLE program has various stages it needs to run to complete a read alignment. Each of these stages will run as a container and will need to execute in a specific order. For example, assuming we already have the indexed multi-reference genome (after running the `mg-ref` stage of BWBBLE and the `index` stage of `mg-aligner`), we would need to run the `align`, `merge`, `aln2sam` and `sampad` stages, in that order.

To instruct Kubernetes on which `Jobs` to run, we will create a custom `AlignJob` resource type and associated controller. When the user submits an `AlignJob` they provide information including the parallelism and reads file to use. The `AlignJob` itself is created using a YAML file, just like a normal Kubernetes `Job`.

```yaml
apiVersion: bwbble.aideen.dev/v1
kind: AlignJob
metadata:
  name: test-job5
  namespace: bwbble-dev
spec:
  alignParallelism: 2
  readsCount: 512000
  readsFile: dummy_reads.fastq
  bubbleFile: chr21_bubble.data
  snpFile: chr21_ref_w_snp_and_bubble.fasta
  bwbbleVersion: "313"
```

Figure 8 Example YAML file to run a BWBBLE alignment job

We will implement a Kubernetes controller to translate the `AlignJob` into a series of Kubernetes `Jobs` which execute each of the BWBBLE stages. This controller runs on the Kubernetes cluster and communicates with the Kubernetes API to perform its duties. This “pull” approach ensures that the controller is not directly accessible from outside the cluster, reducing the attack surface.

By extending the Kubernetes API, we can benefit from its secure authentication, RBAC, auditing and a highly reliable persistence and execution model. This makes it both easier and safer for users to orchestrate DNA alignment jobs.

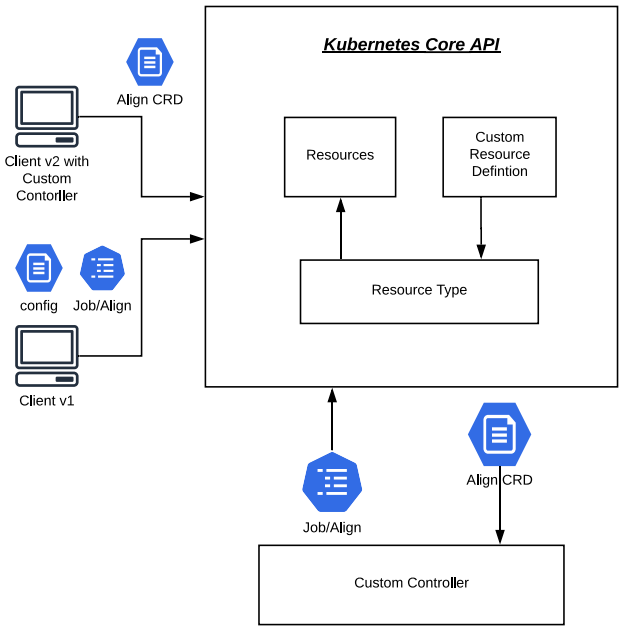

Figure 9 The custom controller translating high level CRDs into custom resource

Once the controller is running within the cluster, an `AlignJob` can be created by the user. The controller will be notified of the creation of this `AlignJob` and coordinate the workflow of running a BWBBLE alignment, ensuring each stage successfully completes before starting the next stage. The controller will continuously run, watching the state of the BWBBLE workflow and the state of the cluster to make sure they are in alignment and that jobs are executing successfully.

The controller allows us to have more control over the workflow of K8s-BWBBLE. We can build a "clean-up" job into Kubernetes that deletes resources such as jobs, pods and config maps after the `AlignJob` completes. It also allows us to integrate benchmarking and data extraction into the controller to expose metrics about the each of the BWBBLE stages that were executed.

We can also use the controller to implement common functionality for BWBBLE workflows. For example, it allows us to calculate the optimum level of parallelism for a given alignment job and could thereby enable users to optimize cost vs performance.

The controller simplifies interaction with our service by eliminating prerequisite Kubernetes knowledge and competence with tools like Helm. Without the controller, tools like Helm are needed to build logic into the YAML files for each BWBBLE stage. With the controller, the user only needs access to `kubectl` or another Kubernetes API client to tell Kubernetes what job they want to run in high-level terms.

3.2.2 Cluster Management

The Kubernetes cluster running the containerized BWBBLE application will be provisioned using Azure Kubernetes Service. This is primarily because the use of Microsoft Azure was a requirement for the project, following AWS-BWBBLE's use of AWS to run BWBBLE on VMs; and Spark-BWBBLE which used Google Cloud Platform.

AKS is Azure's managed Kubernetes offering, providing a fully featured Kubernetes cluster on Azure with no additional setup required. Another significant reason to use AKS is its native integration with `virtual-kubelet` allowing us to use ACIs to supplement the worker nodes in our cluster.

AKS also has integrations with Azure File Share (AFS) and Azure Active Directory. AFS allows multiple `Pods` to have writable storage mounts to the same shared directory which is important when parallelized containers need to write their alignment results to a shared location concurrently.

AKS's integration with AAD provides authentication and authorization for our AKS cluster to better secure the cluster and access to the data stored in AFS.

3.3 Data Storage

The project uses the Azure File Share service offered by Azure Storage to hold the data used by BWBBLE to perform a read alignment. Again, the use of Azure Storage is aligned with the project's requirement to use Microsoft Azure.

Azure File volumes are natively integrated into AKS nodes and supported by Kubernetes on installation of the Azure Disk CSI Driver. The installed driver can be easily swapped out to use EBS volumes provided by AWS or the Google Cloud Engine Persistent Disk volume if running in a different cloud environment. The provider agnostic nature of Kubernetes gives the K8s-BWBBLE user more freedom to choose what resources suit their needs best.

The use of Azure File shares was chosen as the file shares which are accessible by the Server Message Block (SMB) protocol. This enables high concurrency with multiple writes per share. This property is important to enable parallelization of the alignment phase of the program so multiple containers can read sections of the reads file in one file share and output their results to another file share concurrently.

The storage setup in this project uses three file shares – `input`, `ref-output` and `align-output`. The `input` file share stores the data that is needed to create the multi-reference genome and the input reads files that are to be aligned against the constructed multi-reference genome. The `ref-output` file share stores the newly created multi-reference genome and the indexed multi-reference genome.

Once a reads file is sequenced, the `align-output` file share includes the aligned reads that need to be merged and eventually the final output BWBBLE alignment results. All three file shares are mounted to each of the pods created in the AKS cluster and are therefore accessible to the containers within these pods.

In this project's setup, geo-replication is enabled to provide data redundancy with our data stored in both a primary location in North Europe and a secondary failover location in West Europe. A General purpose v2 storage account is used as it provides all the Azure Storage features required with the lowest per-gigabyte capacity prices for Azure Storage.

3.4 Cost Optimization

ACIs are used to run the `align`, `aln2sam` and `sampad` stages of the BWBBLE program. The decision to adapt the containers to use ACIs was made as they are the fastest and cheapest way to run a container in Azure. They provide us the ability to massively parallelize and scale out the alignment phase of BWBBLE, well beyond what is offered by standard Kubernetes nodes.

The virtual node providing ACI integration within AKS advertises support for up to 10k CPUs, 4TB of RAM, 100 GPUs and 5k pods. In practice, the limits are lower and controlled by your Azure subscription. ACIs are suitable for the sort of "burst" work associated with BWBBLE as they can almost instantly execute `Pods` without needing to provision and manage additional worker nodes (VMs).

There is no idle cost associated with using ACIs as we only pay per second of execution time. This enables us to significantly reduce the cost of running an idle cluster, increasing the accessibility of this solution for researchers on limited budgets.

ACIs also provide better security when compared to standard containers which are not considered sufficiently hardened for hostile multi-tenant usage. This is achieved by leveraging hypervisor-level security, ensuring the same isolation as would be provided by a VM.

3.5 Security and Privacy

We need to address the security and privacy considerations of this project as not only does it use Kubernetes; it can also store and process personal genomic data on any third-party cloud provider.

Furthermore, read alignment is the first stage in the sequencing pipeline where sensitive genomic features are observable by an adversary (Lambert, et al., 2018). This visibility is relevant because genomic data can reveal extremely sensitive information about its owner's ancestry, ethnicity and susceptibility to diseases. Genomic data does not necessarily degrade over time and the longevity of security and privacy threats relating to genomic data means extra cause for concern. An adversary

with access to genomic data may be able to derive genomic features that may also be relevant for the owner's descendants and future generations down the line.

If an adversary can align privacy-sensitive reads and identify the genetic variations they carry, as the legitimate BWBBLE program is trying to do, they can perform a trail attack (Malin & Sweeney, 2004). This is a re-identification attack which involves re-connecting the DNA samples to the name and demographics of the person from which they were obtained via trial matching using the REID or REIDIT algorithms (Malin & Sweeney, 2004), (Malin, 2002).

This growing concern surrounding genomic data privacy is presented in a review (Erlich & Narayanan, 2014) which describes how regulatory statutes in the European Union and the United States are now demanding the security and preservation of genomic data privacy. This is extremely relevant at a time when the processing, storage and interpretation of this data often outstrips the compute and storage resources available to most research institutions; causing them to shift workloads to the cloud.

3.5.1 Authentication

The Kubernetes cluster will be configured to use Azure Active Directory (AAD) as the first layer of defence for our system. AAD is an OAuth2 identity and authorization provider which AKS integrates with to provide cluster-level user authentication and access control. This ensures that only authorized users may connect to the K8s API.

AAD includes features like Multi-Factor Authentication and real time threat analysis help reduce the likelihood of a user account being compromised to gain access to the cluster. The AKS cluster was integrated with AAD because Kubernetes doesn't have an available identity management solution to control what users have access to certain resources.

On successful authentication, AAD issues a bearer token to the user which grants `kubectl` (the Kubernetes command line tool) or Azure Portal the ability to impersonate the user and interact with Azure Resource Manager[9] or the Kubernetes cluster, assuming the user has authorization to do so which is verified using RBAC.

3.5.2 Authorization

Once the user has been authenticated, the user will generate an RSA key-pair. This key-pair enables the creation of a certificate, signed by AKS, which may then be used to authenticate the user to the

[9] Azure Resource Manager is the resource management and deployment platform for Microsoft Azure. It is where you can create, delete and update your Azure resources. More information at https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview

cluster. To sign this certificate, the use must first have authenticated via AAD and poses an access token with RBAC privileges granting them access to the Kubernetes cluster.

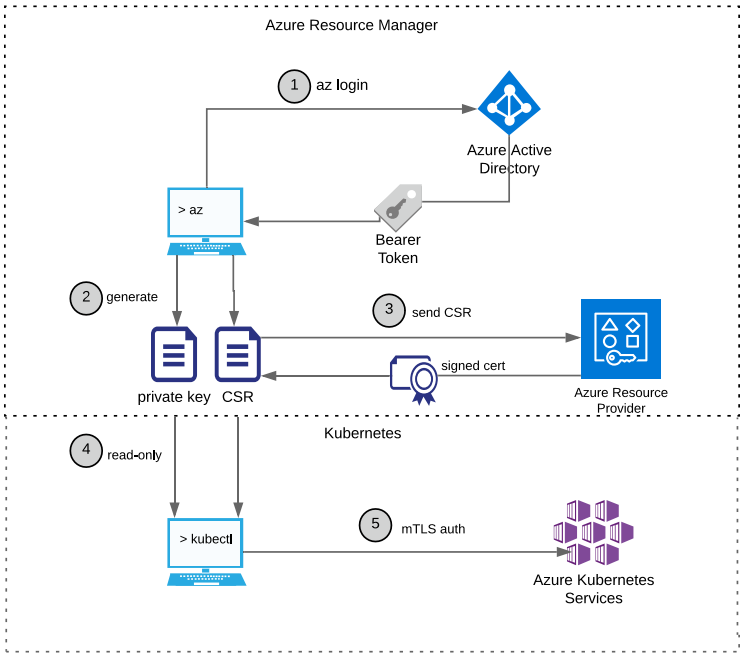

Figure 10 Security Architecture of K8s-BWBBLE

The signed certificate and private key are then stored in the user's `kube_config` file and used by `kubectl` to authenticate the user to the Kubernetes cluster. Once authenticated to the Kubernetes cluster, the Kubernetes RBAC policies enforce access restrictions on Kubernetes resources.

For a user with contributor access to the cluster, this would enable them to now perform actions like `kubectl create pod`.

3.5.3 RBAC

Kubernetes and Azure Role Based Access Control (RBAC) are used for authorization within our cluster to control who has access to the cluster resources. RBAC grants specific permissions to specific user accounts or service accounts. A human accessing the cluster is authenticated as a user account whilst processes inside a `Pod` are authenticated as service accounts.

Permissions sets are defined as `Roles` and these `Roles` are then assigned to user or service accounts through `RoleBindings`. AKS incorporates both Azure RBAC and Kubernetes RBAC mechanisms to provide these `RoleBindings`. `Roles` provide permissions within a specific namespace whilst `ClusterRoles` can grant the same permissions as a `Role` but apply across the entire cluster.

3.5.4 Data Confidentiality

All communication between the user (using `kubectl`) and the Kubernetes API is done over a Mutual TLS (mTLS) connection. This ensures that the identity of both the cluster and user can be cryptographically verified by inspecting their signed X.509 certificates. Additionally, TLS ensures that communications are end-to-end encrypted and authenticated, protecting confidential data from a malicious third-party and ensuring that it is not tampered with.

3.5.5 Isolation

Due to the ability of Kubernetes to run multiple workloads, it is possible that BWBBLE may be deployed alongside other applications. To help minimize the risk that a shared control plane and infrastructure pose, we use Kubernetes namespaces to isolate workloads. Within this project, we use two namespaces: `bwbble-dev` and `bwbble-prod`.

These namespaces enable us to implement administrative and role boundaries as a means of restricting access to BWBBLE and its workloads, preventing unauthorized users from accessing this information. These isolated namespaces also enable us to run different versions of the controller for development and production purposes.

No containers in the cluster are running in privileged mode to reduce the risk of an adversary exploiting the configuration and breaking out of the container, enabling them to run arbitrary code. This sort of privilege escalation also depends on kernel vulnerabilities, mounted network sockets and filesystems.

4 Implementation

4.1 Phase 1: Prototyping Docker

Before we run BWBBLE on Kubernetes, we need to containerize the application. We created a Dockerfile for `mg-ref` (see Appendix 1) and another for `mg-aligner` (see Appendix 2). Within these Dockerfiles are a series of instructions describing how Docker should build the container image.

The `mg-ref` and `mg-aligner` container use a multi-stage build to keep our container image and container as lightweight as possible to increase the likelihood of achieving a linear speedup. The first stage of both containers is the exact same. They both use Ubuntu as the base image and install all the dependencies they need to build BWBBLE's executables. These include `g++`, `gcc`[10], `make` and `zlib1g-dev`. Builds are conducted the same way one would locally, by running `make all`.

The second stage of both containers is also based on the Ubuntu base image. We copy the compiled executables from the previous stage and install any runtime dependencies like `libgomp1`. In addition to the compiled executables, we add an `entrypoint.sh` script (see Appendix 3 and Appendix 4) which acts as the starting point for the container during execution. This entrypoint script checks the `/var/run/container_args` file for a sequence of command line arguments. If there is a such a file present these arguments supersede those provided to the container at start-up. This enables us to run the containers in the Azure Container Instances environment, which currently does not support passing command line arguments to containers directly[11].

We then built the Docker images, using the `-t` flag to provide a human readable alias for the image version.

```
docker build -t bwbble/mg-ref:latest
docker build -t bwbble/mg-aligner:latest
```

These Docker images can then be run on any Linux based Docker host, or on a Windows host using Docker Desktop.

```
docker run -it --rm bwbble/mg-ref:latest
docker run -it --rm bwbble/mg-aligner:latest
```

[10] g++ and gcc are components of the GNU Compiler Collection responsible for the compilation of C++ and C source code, respectively.

[11] Azure Container Instances, when used with AKS, do not support the passing of command line arguments at the time of writing (https://docs.microsoft.com/bs-latn-ba/azure/aks/virtual-nodes-portal#known-limitations).

4.2 Phase 2: Prototyping Kubernetes

Once the BWBBLE application was containerized, the next stage was to run BWBBLE on Kubernetes. I used the Kubernetes cluster on my local machine, provided by Docker Desktop, to do this.

To run BWBBLE on Kubernetes, I created a Job definition for each BWBBLE stage: `dataprep`, `comb`, `index`, `align`, `aln2sam` and `sampad`. These job definitions instruct Kubernetes to run a pod with the correct BWBBLE container and arguments. The Job definitions themselves are represented as a series of YAML files and provide Kubernetes with all the information it needs to create and run each container.

To simplify the use of these YAML files, I used a tool called Helm which enables the use of template variables within these files. The collection of templatized YAML files is referred to as a Helm chart. These Helm charts orchestrate all the templates in a way that enables us to run a full BWBBLE alignment – from `dataprep` to the `sampad` stage.

```
helm create bwbble
```

Helm's templates allow you to provide a set of default values in a `values.yaml` file (see Appendix 5). The values within this file may then be accessed within any of the YAML templates, ensuring consistency. For example, each running Kubernetes Job needs to know where its data files should be stored; using the `values.yaml` file we can ensure that they all use the correct directories.

To access the required data files, Kubernetes `hostPath` volume mounts were used. These `hostPath` volumes expose the specified directories from our host filesystem into the pods running on the cluster. The volumes mounted included `input`, `mg-aligner-output` and `mg-ref-output`.

To control the BWBBLE workflow, ensuring that each stage waited for the prior stage to complete before proceeding, I used init containers. Init containers are specialized containers that run before the main application container runs. They enable us to delay the starting of a stage until the preceding stage has completed. To do this, the `bitnami/kubectl` image is used to access the Kubernetes CLI. The `kubectl wait` command causes the init container to wait until a specific job has completed before exiting.

```
kubectl wait --for="condition=complete" job/aln2sam --timeout=-1s
```

To deploy the Helm chart onto the Kubernetes cluster, the Helm CLI was used as shown below.

```
helm install example-release ./bwbble
```

4.3 Phase 3: Prototyping the Parallelization

To parallelize the `align` stage, we spit up the `reads.fastq` file to enable several BWBBLE instances to process it simultaneously. Each BWBBLE instance can then map a section of the reads file against the multi-reference genome. Another option we considered was sending pointers to locations in the reads file to the containers. The containers could then process this small section.

```
@21_16211141_16211694_4:0:0_0:0:0_1/1
TGTTTCTTTTCTTTAGGTCAGAGCCCATTAAATGATTCACTTTATGTATATCACTTAGACAATAAACATTCTATCCCTTTCCTCTCTGACTGTAAGACCA
+
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
@21_32773354_32773803_3:0:0_0:0:0_2/1
CATGGCTGCCATATCCCAATAGAGGAACAGTTTACCATTGTTTATATTCGGCTCTAGGCTACAATAGGAGGGCAGGGCCCATTTTTTCTTGCCTAGTGTT
+
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
@21_24684513_24684883_4:0:0_1:0:0_3/1
GTACAGAACTGAATTTCTAAGGAAGACTATGAGTCCTCCATAGACTTCCCTGGAATTTACTCCATTAGACAAAATGTTATGGCGGCAGATCAATCTGAGA
+
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
```

Figure 11 Sample reads file used in read alignment

Figure 11 shows a section of a standard reads file used in read alignment programs. Each sequence in the reads file consists of four lines[12]. To ensure that BWBBLE can correctly align these reads, it is important that we divide them into valid chunks along this four-line boundary. At this point in the project, these chunk ranges are specified manually. At a later stage in the project, we will use the controller to automatically generate these file ranges for us given a requested degree of parallelism provided by the user.

To support aligning a subset of the `reads.fastq` file, we modified the BWBBLE alignment code as shown in Appendix 6. These changes introduced a "skip count" (`-s`) flag and a "process count" (`-p`) flag. The skip count flag specifies the number of lines to skip in the reads file before BWBBLE begins processing. By default, this is set to `0` which means it will start at the beginning of the reads file without skipping any lines. The process count flag specifies the number of lines to process in the reads file. By default, this is set to `-1` which will process the entire reads file, stopping when the `EOF` marker is reached.

[12] The first line is a sequence identifier which shows information about the sequencing run, the second line holds the read sequence, the third line is the (+) separator and the fourth line includes sequence quality scores which measure the probability of a base being identified incorrectly.

```
Usage: bwbble align [options] <seq_fasta> <reads_fastq> <output_aln>
Options: M    mismatch penalty (default: 3)
         O    gap open penalty (default: 11)
         E    gap extend penalty (default: 4)
         n    maximum number of differences in the alignment (gaps and mismatches) (default: 0)
         l    length of the seed (seed := first seed_length chars of the read) (default: 32)
         k    maximum number of differences in the seed (default: 2)
         o    maximum number of gap opens (default: 1)
         e    maximum number of gap extends (default: 6)
         t    run multi-threaded with t threads (default: 1)
         s    the number of lines to skip in the read file before processing (default: 0)
         p    the number of lines to process in the read file (default: -1)
         S    align with a single-genome reference
         P    use pre-calculated partial alignment results
```

Figure 12 Extended BWBBLE align option list to support alignment parallelization

The `align` Job template within the Helm chart was modified to enable a parallelized read alignment by creating multiple Job instances and passing the skip count and process count flags to their containers (see `bwbble/templates/align.yaml` in Appendix 17).

With multiple `align` Jobs running in parallel, the result is multiple alignment files. These files need to be merged into a single file. This task is performed by a `merge` Job (see Appendix 7), which uses the `busybox` container image and simply concatenates the parallelized alignment results in the `mg-align-output` directory into a single alignment results file. This `aligned_reads.aln` file is then written to the `mg-align-output` directory for further processing.

```
cat aligned_reads.0.aln aligned_reads.1.aln > aligned_reads.aln
```

Basic benchmarking was conducted at this point to compare the time taken to run a parallelized alignment running on Kubernetes with the time taken to run an alignment on BWBBLE.

4.4 Phase 4: Prototyping Azure

The next stage of the project was to run the containerized version of BWBBLE on Azure Kubernetes Service (AKS). Doing so enables us to run at much higher levels of parallelism, taking advantage of Azure's scale-out capacity. AKS also has native integration for Azure File Shares which we use to store the data used and produced by BWBBLE. Rounding out the offering, AKS's integration with Azure Active Directory enabled us to increase the security posture of our project by leveraging an industry leading authentication and authorization platform.

4.4.1 Deploying an Azure Kubernetes cluster

To configure a Kubernetes cluster on Azure, we were required to create an Azure account. Within this Azure account we created a resource group – Azure's means of logically grouping related resources – and deployed an Azure Storage Account and Azure Kubernetes Service cluster within it. Importantly,

we ensured that both the AKS cluster and Azure Storage account were created in the same region to minimize network latency and improve performance.

The creation of the AKS cluster can be performed using the Azure CLI, specifying the resource group name, the cluster name and the number of nodes.

```
az aks create
    --resource-group myResourceGroup
    --name myAKSCluster
    --node-count 1
    --generate-ssh-keys
```

Once the AKS cluster was created, we needed to configure `kubectl` to connect to the cluster. To do this we used the Azure CLI to generate a set of credentials for the cluster and place them in our `kube_config` file.

```
az aks get-credentials
    --resource-group myResourceGroup
    –-name myAKSCluster
```

4.4.2 Setting up Azure File Storage

To minimize cost and maximize performance, we used a General Purpose v2 storage account. This tier provides us with support for Azure File Storage along with the lowest per-gigabyte capacity prices for Azure Storage. We enabled Geo-replication to provide data redundancy with our data stored in both a primary location in North Europe and a secondary failover location in West Europe.

We used a series of Azure File shares to hold the data that BWBBLE operates on. The storage setup in this project uses three file shares – `input`, `ref-output` and `align-output`. The `input` file share stores the data that is needed to create the multi-reference genome and the input reads files that are to be aligned against the constructed multi-reference genome. The `ref-output` file share stores the newly created multi-reference genome and the indexed multi-reference genome. Once a reads file is sequenced, the `align-output` file share includes the aligned reads that need to be merged and eventually the final output BWBBLE alignment results.

Kubernetes can access Azure File Shares through the `azureFile` volume driver, which replaces the `hostPath` volume driver used previously. As a result of using Helm, switching to the `azureFile` volume driver simply involved creating an `aks.values.yaml` (see Appendix 8) which used Azure File shares for storage instead of the `hostPath` volume mounts (see Figure 13).

```
volumes:
  input:
    azureFile:
      secretName: azure-secret
      shareName: input
      readonly: false
  refoutput:
    azureFile:
      secretName: azure-secret
      shareName: ref-output
      readonly: false
  alignoutput:
    azureFile:
      secretName: azure-secret
      shareName: align-output
      readonly: false
```

Figure 13 Mounting Azure File shares in `aks.values.yaml`

For security reasons, Azure File shares require you to authenticate using a secure access key. These access keys need to be provided to our Kubernetes cluster in the form of Kubernetes secrets. These secrets are stored in etcd, where they are encrypted at rest.

The format of this secret is important as the `azureFile` volume driver relies on it having specific fields. Specifically, it is required to contain both the `azurestorageaccountname` and `azurestorageaccountkey` fields. We then reference this secret in the `azureFile` volume definitions as shown in Figure 13. At this point, we can then use a Kubernetes `volumeMount` to expose the storage volume within our containers.

4.5 Phase 5: Prototyping the Controller using the Python Kubernetes client

In this phase, we replaced the use of Helm with a `controller.py` script (see Appendix 9) which uses the `python-kubernetes` client to create the Kubernetes Jobs and manage the alignment workflow. Doing so enabled us to remove the need for init containers and move a step closer to the final controller design.

4.5.1 Using the Kubernetes Client

The `python-kubernetes` client can be configured to use either your local `kube_config` for authentication, or Service Principal credentials when running in a Kubernetes pod. Switching between the two is extremely simple, requiring us to call the `config.load_kube_config()` method or `config.load_incluster_config()` respectively.

Use of the Kubernetes APIs is then facilitated through a series of wrappers, including the `BatchV1Api` wrapper shown in Figure 14. These wrappers make it possible to manage resources belonging to a

specific Kubernetes API group – the `batch/v1` API group in this case. A full list of the methods available for use with the `batch/v1` API is visible in Figure 15. Other API wrappers provide similar methods.

```
config.load_kube_config()
configuration = Configuration()
api_client = BatchV1Api(ApiClient(configuration))
```

Figure 14 Setting up the Kubernetes client configuration

Method	HTTP request
create_namespaced_job	**POST** /apis/batch/v1/namespaces/{namespace}/jobs
delete_collection_namespaced_job	**DELETE** /apis/batch/v1/namespaces/{namespace}/jobs
delete_namespaced_job	**DELETE** /apis/batch/v1/namespaces/{namespace}/jobs/{name}
get_api_resources	**GET** /apis/batch/v1/
list_job_for_all_namespaces	**GET** /apis/batch/v1/jobs
list_namespaced_job	**GET** /apis/batch/v1/namespaces/{namespace}/jobs
patch_namespaced_job	**PATCH** /apis/batch/v1/namespaces/{namespace}/jobs/{name}
patch_namespaced_job_status	**PATCH** /apis/batch/v1/namespaces/{namespace}/jobs/{name}/status
read_namespaced_job	**GET** /apis/batch/v1/namespaces/{namespace}/jobs/{name}
read_namespaced_job_status	**GET** /apis/batch/v1/namespaces/{namespace}/jobs/{name}/status
replace_namespaced_job	**PUT** /apis/batch/v1/namespaces/{namespace}/jobs/{name}
replace_namespaced_job_status	**PUT** /apis/batch/v1/namespaces/{namespace}/jobs/{name}/status

Figure 15 Methods exposed by the BatchV1Api

4.5.2 Implementing the Workflow

Within the `controller.py` script, we defined a series of methods to run each of the BWBBLE stages associated with a full alignment. These methods leverage a common `create_job_resources` method (see Appendix 9) to create the relevant Kubernetes jobs; as well as a common `wait_for_all_jobs` method which allowed us to wait for jobs to complete before proceeding.

This enabled us to coordinate a full alignment workflow by calling these methods in the following order:

1. `run_data_prep()`
2. `run_comb()`
3. `run_index()`
4. `run_align()`
5. `run_merge()`

6. `run_aln2sam()`
7. `run_sam_pad()`

To make it possible to run multiple workflows, a unique release name was generated using the current timestamp and included in the names of every Kubernetes resource created during the workflow. This was handled within the `create_job_resources` method and would result in resource names following the format `bwbble-$release-$stage` (for example: `bwbble-2020-02-17-14-23-comb`).

With Kubernetes requiring that every resource has a unique name, we also needed to add unique suffixes to parallelizable jobs like `align`. This was facilitated with a `name_suffix`, allowing us to create jobs like `bwbble-2020-02-17-14-23-align-0`.

We also found that some stages, like `index`, required additional resource allocations to run successfully. For `index` operations we found that the process required memory equal to about 3x the base file size. To facilitate this, the `create_job_resources` method was extended to allow a `V1ResourceRequirements` specification to be passed to it. This specification informs Kubernetes that the job requires a specific amount of CPU and Memory to run, allowing the cluster to allocate the required resources.

4.5.3 Waiting for Dependencies

Once a Kubernetes Job for a BWBBLE stage is created, we must wait for this Job to complete before proceeding onto the next BWBBLE stage. To do this use the `wait_for_all_jobs` method we created to replace the init containers used previously in the Helm charts.

The `wait_for_all_jobs` method uses Kubernetes' `watch` method to receive a stream of updates to the status of the Kubernetes Job. When the Kubernetes Job status is populated with a completion time, we know the Job has been completed and we can move onto the next BWBBLE stage. The code implementing this functionality is visible in Figure 16.

```python
def wait_for_all_jobs(
    namespace: str, release: str, stage: str, resources: List[kubernetes.client.V1Job]
):
    watcher = kubernetes.watch.Watch()

    pending_jobs = set([r.metadata.name for r in resources])

    for event in watcher.stream(
        kubernetes.client.BatchV1Api(api_client).list_namespaced_job,
        namespace,
        label_selector=f"bwbble-release={release},bwbble-stage={stage}",
    ):

        if event["object"].status.completion_time:
            pending_jobs.remove(event["object"].metadata.name)

            if len(pending_jobs) == 0:
                watcher.stop()
                return
```

Figure 16 Kubernetes Watcher checking for updates to the running Job.

4.6 Phase 6: Using Azure Container Instances

Azure Container Instances (ACIs) allow us to take advantage of per-second billing and a massive degree of burst scalability to run more complex workflows. Updating the `controller.py` script to use ACIs primarily involved making changes to the Kubernetes Job specification.

4.6.1 Passing Command Line Arguments to ACIs

There are also some limitations to ACIs – the most notable of which is the inability to pass command line arguments to the containers. To work around this, we create a `ConfigMap` with the arguments contained within it. This `ConfigMap` can then be mounted into the ACI through the `ConfigMapVolume` type, which exposes keys within the `ConfigMap` as files on the filesystem. This causes the arguments to become accessible at `/var/run/args/container_args` within the container. On starting the container, the `entrypoint.sh` script will check this location to see if there are any arguments defined and will pass them to BWBBLE.

We automated process of creating this ConfigMap and linking it to the container by extending the `create_job_resources` with a new `use_config_map_args` parameter as shown in Figure 17.

```
if use_aci and not use_config_map_args:
    raise Exception(
        "Azure Container Instances require that arguments are passed using a file"
    )

if use_config_map_args:
    env_array.append(
        client.V1EnvVar(name="ARGS_FILE", value="/var/run/args/container_args")
    )

    created_resources.append(
        kubernetes.client.CoreV1Api(api_client).create_namespaced_config_map(
            namespace,
            kubernetes.client.V1ConfigMap(
                api_version="v1",
                kind="ConfigMap",
                metadata=client.V1ObjectMeta(
                    name=f"bwbble-{release}-{stage}{name_suffix}", labels=labels
                ),
                data={
                    "container_args": " ".join(
                        [f"'{a}'" if " " in a else a for a in args]
                    ),
                },
            ),
        )
    )
```

Figure 17 Creating the ConfigMap to pass arguments to the ACI

4.6.2 Alignment Ranges

With the ability to easily create many parallelized align jobs, we created a `Range` class to automate the naming and splitting of a FASTQ file. The Range object has a `start` position and `length`. The `start` property corresponds to the "skip count" flag we added to the BWBBLE align command while the `length` corresponds to the "process count" flag. These flags control the portion of the `reads.fastq` file while will be processed by a specific BWBBLE instance.

To simplify the generation of these ranges for a given `reads.fastq` file, we implemented a `generate` method. This method accepts a user supplied level of parallelism and the number of reads in the `reads.fastq` file. The reads count is divided by the level of parallelism and the chunks of valid file ranges are generated. The controller will iterate through this list of file ranges and will run alignment jobs with the specified start and length position.

Finally, we use a `name` property to standardize the way that files and jobs are named for a given range. This is passed to the `create_job_resources` method as the `name_suffix` parameter.

```python
class Range(object):
    def __init__(self, start: int, length: int = -1):
        self.start = start
        self.length = length

    @property
    def name(self) -> str:
        if self.length == -1:
            return f"{self.start}-end"
        return f"{self.start}-{self.start+self.length}"

    @staticmethod
    def generate(num_reads: int, parallelism: int = 1):
        rrange = floor(num_reads/parallelism)
        ranges = []
        for i in range(0, parallelism):
            ranges.append(Range(i * rrange, rrange))

        ranges[len(ranges)-1].length = -1

        return ranges
```

Figure 18 Range class responsible for generating file ranges to split the FASTQ file

At this stage, we ran several benchmarks to confirm that the controller worked correctly and we saw the expected linear reduction in alignment time as we increased the degree of parallelism. We took a pair of measurements for this purpose: the Kubernetes job lifespan (measured from when the job was started until it was completed) as well as the time reported by BWBBLE's `align` job internally.

To simplify the extraction of this information, we created an `execution_times` method (see Appendix 9) which queried the Kubernetes API and exported a summary of this information.

4.7 Phase 7: Kubernetes Controller

Finally, we focussed on converting the `controller.py` script into a true Kubernetes controller. A Kubernetes controller enables users to interact directly with the Kubernetes API through custom resource types. This removes need for the user to run a local application to coordinate their workloads. These custom resources are registered with Kubernetes in the form of a Custom Resource Definition (CRD). The CRD used by this project was called a `V1AlignJob` (see Appendix 12). The `V1AlignJob` represents a full BWBBLE alignment using a series of simple parameters to control the workflow.

4.7.1 Custom Resource Definition

The `V1AlignJob` follows the standard resource format used by Kubernetes. In addition to the core Kubernetes resource properties (`apiVersion`, `kind`, `metadata`) we define a `spec` and `status` field for the `V1AlignJob`. Within `spec` we configure properties like `readsFile`, `bwbbleVersion`, `readsCount`, `alignParallelism`, `bubbleFile` and `snpFile`. The `status` field holds runtime information used by the controller to describe the status of the job. The properties here include a `startTime`, `endtime`,

stage, waiting_for and execution_times. The Job startTime and endTime enable us to derive the full runtime for the V1AlignJob while the stage field identifies what BWBBLE stage the V1AlignJob is currently running.

4.7.2 Monitoring Changes

The controller functions by monitoring the Kubernetes API for the creation of new V1AlignJobs. When a V1AlignJob is created, the controller is responsible for following a sequence of steps to transition from the start to the end of the workflow. To achieve this, the controller uses a pair of Kubernetes watchers to observe changes to V1BatchJobs as it waits for them to complete, as well as V1AlignJobs which may be created or modified by both the user and controller.

The BatchJobWatcherThread monitors V1BatchJobs created by create_job_resources and removes them from the waiting_for set for the associated V1AlignJob. The removal of these jobs from the waiting_for set triggers an update event for the V1AlignJob which will be processed by the AlignJobWatcherThread.

The AlignJobWatcherThread is responsible for identifying changes made to V1AlignJobs, including their creation through kubectl as well as any changes made by the BatchJobWatcherThread. When a V1AlignJob has no outstanding waiting_for jobs, the next stage will be started.

To make it easier to implement this functionality, we refactored the controller.py script to treat each stage in a uniform manner – implementing their logic in distinct classes. Each of these stages is responsible for dispatching Kubernetes V1BatchJobs to execute a specific command. The unique names of these V1BatchJobs are then added to a waiting_for set, allowing the controller to keep track of ongoing Jobs and identify when to progress to the next stage.

4.7.3 Execution Times

To simplify benchmarking, the V1AlignJob has a status.execution_times field which holds information on the amount of time spent executing each stage of the workflow. These execution times are populated immediately after the completion of each BWBBLE stage and capture the total time taken for the stage to execute, as well as the time reported by the container itself (internal) in some cases. An example of this is shown in Figure 19.

```
status:
  endTime: 2020-04-12T19:41:33.264842
  executionTimes:
    align:
      bwbble-bench-noaci-p3-r1-align-0-170666:
        internal: "0:00:40.600000"
        total: "0:01:13"
      bwbble-bench-noaci-p3-r1-align-170666-341332:
        internal: "0:00:39.720000"
        total: "0:01:13"
      bwbble-bench-noaci-p3-r1-align-341332-end:
        internal: "0:00:38.210000"
        total: "0:01:18"
    aln2sam:
      bwbble-bench-noaci-p3-r1-aln2sam:
        total: "0:00:37"
    cleanup: {}
    merge:
      bwbble-bench-noaci-p3-r1-merge:
        total: "0:00:03"
    sampad:
      bwbble-bench-noaci-p3-r1-sampad:
        total: "0:00:25"
  stage: completed
  startTime: 2020-04-12T19:39:08.261194
  waitingFor: []
```


Figure 19 Captured final "Status" output from an AlignJob

To calculate the `total` time for a stage, we compare the `startTime` and `endTime` reported by each Kubernetes Job – reporting the difference. This information is reported for each distinct stage and job within the `V1AlignJob`, providing a high level of detail for comparisons and benchmarking. The `internal` time is extracted from a container's logs directly, allowing BWBBLE to measure the time spent in individual portions of its code for better accuracy. By comparing the `total` and `internal` time, we can determine the overhead of running BWBBLE in a containerized environment.

The logic to gather these times (see Appendix 10) is very similar to the logic used with the `controller.py` script, however it is implemented internally as a special stage type within the controller – greatly simplifying integration with the rest of the workflow.

These fields allow a user to quickly access useful metrics about their `V1AlignJob` using `kubectl`. This information can be used for benchmarking, or just basic status reporting. To make finding this information easier, we have provided several aliases for the job data: `alignjob`, `alignjobs`, `aj` or `ajs`.

```
kubectl get alignjobs --namespace bwbble-dev
```

The implementation of the `status` and `spec` fields of the `V1AlignJob` can be seen in the `job_spec.py` file (see Appendix 12).

4.7.4 Clean-up

To ensure that native Kubernetes resources are cleaned up after a `V1AlignJob` completes, we added a `cleanup_job` stage to the workflow. This stage executes after all the BWBBLE stages have completed and deletes all the Kubernetes `Jobs` and `ConfigMaps` that were created as part of the `V1AlignJob`.

When performing the clean-up operation (shown in Figure 20) it is important to use the foreground propagation policy. This policy ensures that child resources (`V1Pods` in the case of a `V1BatchJob`) are also deleted. This matches the default behaviour of `kubectl` when deleting jobs.

It is important to perform this clean-up as ACIs are not removed until the pod which owns them is deleted. This build-up of ACIs can quickly lead to an inability to schedule new pods due to Azure subscription limits on the number of ACIs which may be created.

```
class CleanupJob(Job):
    def __init__(self):
        super().__init__("cleanup")

    def run(self, job: V1AlignJob):
        self.api_instance.delete_collection_namespaced_job(
            job.metadata.namespace,
            label_selector=f"bwbble-alignjob-name={job.metadata.name}",
            propagation_policy="Foreground",
        )

        CoreV1Api(self.api_client).delete_collection_namespaced_config_map(
            job.metadata.namespace,
            label_selector=f"bwbble-alignjob-name={job.metadata.name}",
        )
```

Figure 20 Clean-up Job that deletes all Kubernetes resources for a V1AlignJob

4.7.5 Running an AlignJob

Running an alignment workflow is done by submitting a `V1AlignJob` resource definition to the Kubernetes. The controller's `AlignJobWatcherThread` will receive a change event for the new resource and attempt to transition this job into its final (`completed`) state through a series of state transitions (shown in Figure 21).

Figure 21 State transitions used by a V1AlignJob

The `on_align_job_updated` method handles the change in state in an `AlignJob`. It first checks if the `V1AlignJob` is waiting for any `V1BatchJobs` to complete within the stage. In the case of a parallelized align stage, this may involve waiting for many `V1BatchJobs` to complete.

When all jobs for a given stage have completed, the controller will execute the code for the next stage and update the `V1AlignJob` with information about the new stage, including which `V1BatchJobs` it is waiting on.

The workflow will continue to move forward with the watcher threads continuously checking for changes in states in the `V1AlignJob` and `V1BatchJobs` and updating the `V1AlignJob`'s state to reflect these changes until it reaches the `complete` state.

By storing information about the current `stage` and the list of `V1BatchJobs` that we are `waiting_for` within the `V1AlignJob` itself, we enable the controller itself to be stateless – reducing the risk associated with having a single coordinator in this system.

4.8 Phase 8: Final Deployment

While it is possible to run the controller on your laptop and orchestrate workflows on AKS through the Kubernetes API, one of the goals of this project is to remove the need for custom software or skills on behalf of the user.

To remove the need to run the controller locally, we can deploy it on the Kubernetes cluster and then orchestrate workflows using only the Kubernetes API. This means that a user only needs access to `kubectl`, however it is possible to build a user interface which takes advantage of the Kubernetes API to further simplify this.

We used the Kubernetes `V1Deployment` resource to run the controller on the cluster. This resource makes sure that a specific version of the controller is running at any given time and simplifies updating it to a newer version.

4.8.1 Authentication and Authorization

With the controller is running on the cluster, it will no longer have access to our laptop's `kube_config` file, meaning is will need to rely on the service account assigned to its Pod to authenticate to the Kubernetes API. The `default` service account does not have any role bindings associated with it, preventing it from accessing any dangerous methods in the Kubernetes API. This will prevent the controller from being able to create and update `Jobs`, `ConfigMaps` and our `V1AlignJob` custom resources.

To solve this, we need to use a combination of role-bindings, an RBAC role and a custom service account to authenticate the controller. These are managed through the creation of Kubernetes resources as can be seen in Appendix 14.

We create a dedicated `bwbble-controller` service account and configure the controller's Pod to use this service account instead of the `default` one. We then create a Role and configure it to allow access to `V1AlignJobs`, `ConfigMaps`, `Jobs`, `Pods` and any other APIs the controller needs access to. This role is then associated with the service account through a `RoleBinding`.

4.8.2 Deployment Steps

The deployment itself is performed by using `kubectl` to upload the `deployment.yaml` file (see Appendix 14) to the cluster. This will update any resources which already exist and create resources which are missing.

```
kubectl apply -f deployment.yaml
```

Within this `deployment.yaml` file are the individual Kubernetes resources describing the `V1Deployment`, `V1ServiceAccount`, `V1Role` and `V1RoleBinding`. By keeping them in a single file, we greatly simplify the process of configuring a Kubernetes cluster for use with BWBBLE.

4.8.3 User Experience

The user submits an AlignJob in the form of a YAML file. This is significantly more straightforward than managing multiple Helm templates as required by the previous iteration of the project. The reads file will be parallelized and split using the read count and parallelism specified by the user.

```yaml
apiVersion: bwbble.aideen.dev/v1
kind: AlignJob
metadata:
  name: test-job5
  namespace: bwbble-dev
spec:
  alignParallelism: 2
  readsCount: 512000
  readsFile: reads.fastq
  bubbleFile: bubble_file.data
  snpFile: multi_ref_genome.fasta
  bwbbleVersion: "313"
```

Figure 22 Sample Align Job submitted by our user

The user submits this high-level alignment job to run on Kubernetes.

```
kubectl apply -f <yaml_file>
```

The controller runs continuously when started using the `main.py` file (see Appendix 13). The controller is stateless, keeping state in `etcd`. This means if you execute the same job twice, using the same reads file and multi-reference genome, the alignment results will be received immediately.

5 Benchmarking

Benchmarking of K8s-BWBBLE was undertaken by submitting an `AlignJob` to the controller with 15 different degrees of parallelism. We automated this process using the `benchmark.py` script (see Appendix 15), running 5 alignment jobs for each parallelism measurement to increase accuracy.

The Kubernetes pods running `mg-aligner` were each allocated 1 vCPU and 2GB of memory, roughly the equivalent of a small virtual machine. All benchmark runs were conducted with the same pre-indexed chromosome 21 multi-reference genome and a reads file consisting of 512,000 reads.

This size reads file is sufficient for prototyping the performance of this project but in practice a much larger reads file is used. We synthesized our own reads file of 5,632,000 reads by concatenating the smaller reads file we had. However, due to a crash in BWBBLE when using this file, we were unable to complete an alignment. The objective of the project was not to optimize BWBBLE, so we continued with the smaller reads file.

To minimize the effect of outliers on the overall execution time, we use the median for all measurements. These outliers are usually `Pods` that failed and were rescheduled by Kubernetes during the `AlignJob`. `Pods` can fail for several reasons including insufficient resources or an inability to pull the container image. The median execution time displayed for these graphs represents the median execution time for 5 identical `AlignJobs` run with the same level of parallelism.

The cost for running all the benchmarks using ACIs amounted to €4.28. This is extremely low considering that each alignment uses several ACIs and we ran 5 alignments for each of the 15 degrees of parallelism benchmarked. This brings the cost for alignment to €0.11 per million reads, excluding storage. To cost of storing the genomic data used in our benchmarking over the course of two months amounted to €8.06.

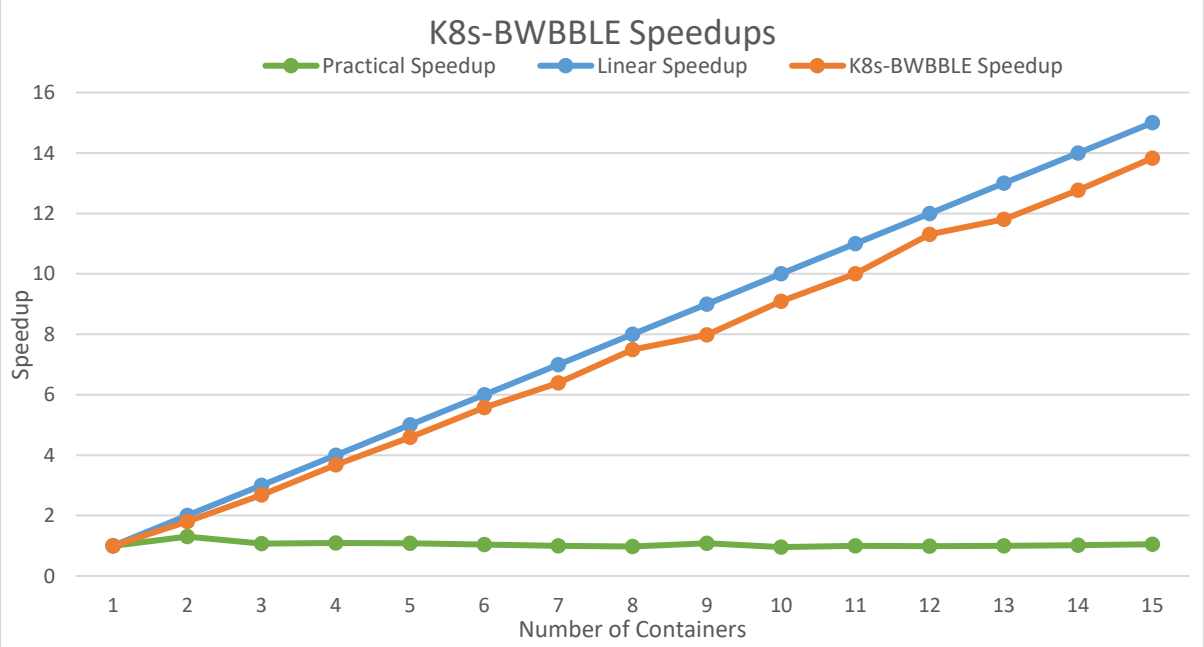

Figure 23 Speedups achieved by K8s-BWBBLE

Figure 23 plots a theoretical perfectly linear speedup, K8s-BWBBLE Align Time speedup and practical speedup against the level of parallelism. K8s-BWBBLE Align Time refers to the time to execute the BWBBLE align code, excluding the time taken to start and schedule its `Pod`. The practical speedup is the end-end time for completing an `AlignJob` using K8s-BWBBLE, from submitting the `AlignJob` to deleting all the resources used.

The average speedup factor of K8s-BWBBLE is 91%. This means by adding an extra container we will reduce the align stage time by $\frac{91}{N}\%$, where N is the number of containers.

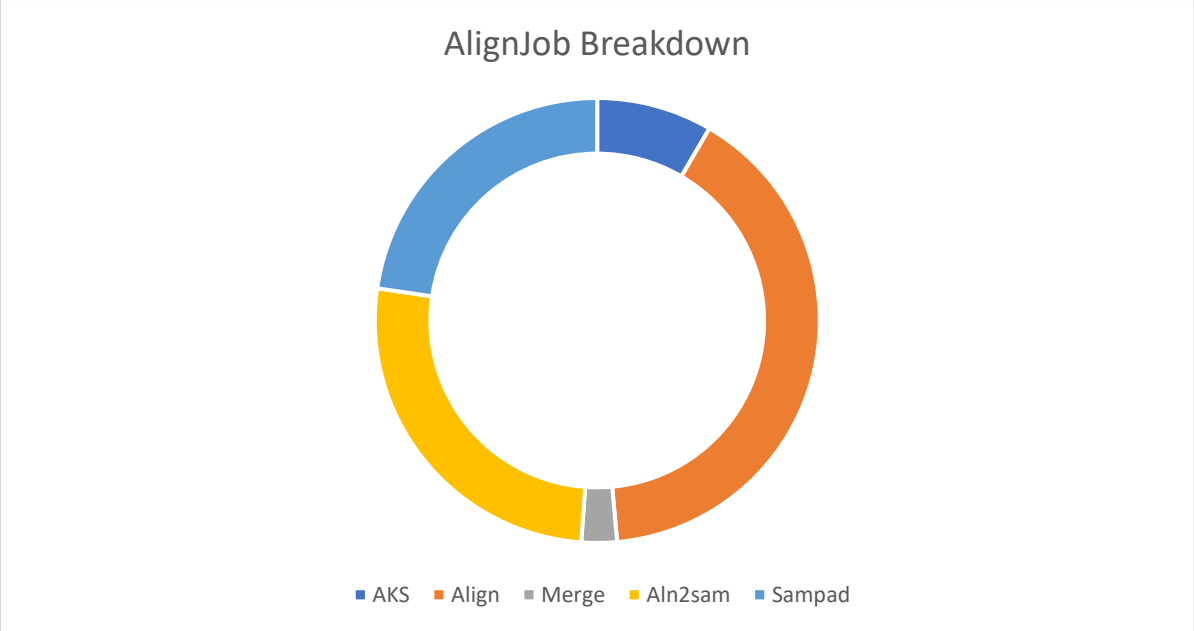

Figure 24 Breakdown of where we spend our time when completing a full alignment

The practical speedup, however, does not benefit as much from increased levels of parallelism. This is because the practical speedup is dominated by time spent in other stages of BWBBLE and the AKS overhead. Although the `align` stage is the longest individual BWBBLE stage, the other stages combined make up most of our experienced end-to-end run time as can be seen in Figure 24. With a larger reads file, it is expected that the practical speedup will reduce more noticeably relative to the degree of parallelism.

Figure 24 shows that on average we spend 40% of our time completing the `align` stage, 3% completing the `merge` stage, 26% on the `aln2sam` stage, 23% completing `sampad`. We spend 8% of our time on processes relating to AKS. This includes the scheudling of containers, communication and deleting of resources.

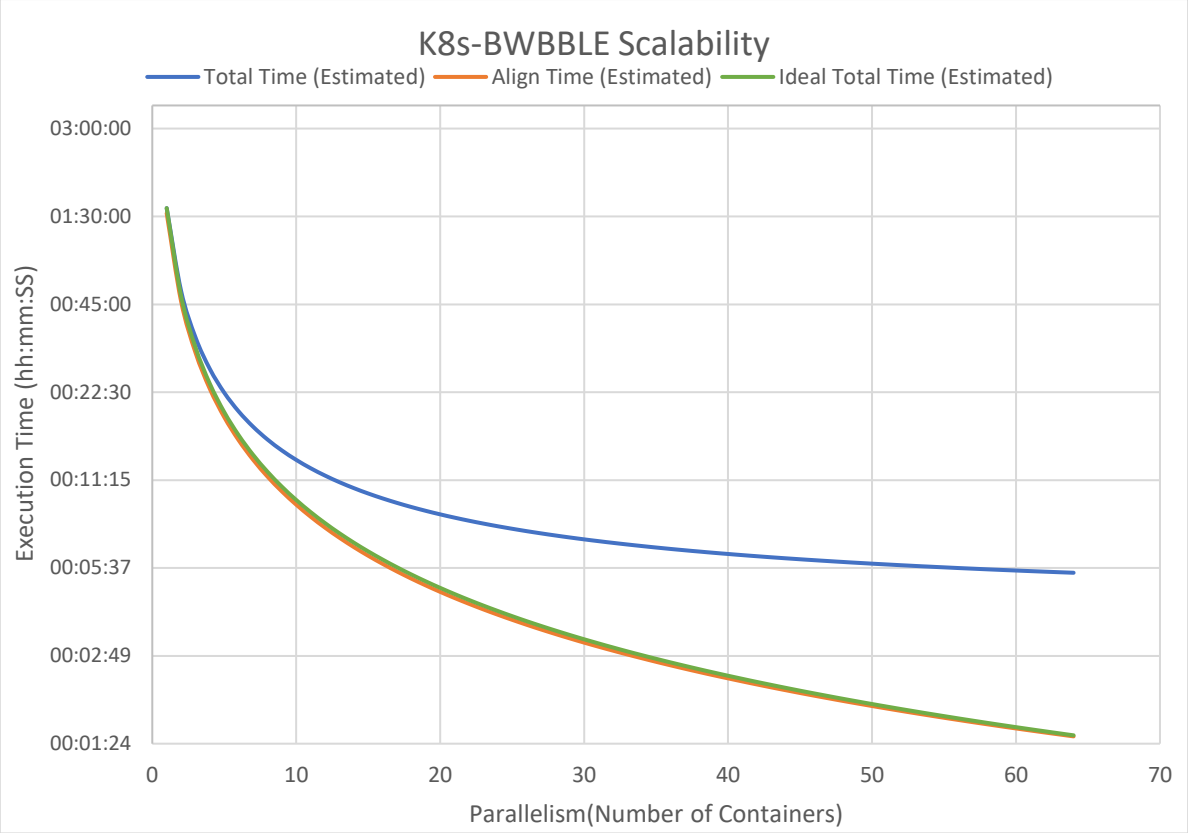

Figure 25 Theoretical Speedup for a large reads file

To reduce the practical runtime of the program we need to reduce the time taken to complete the other stages. To demonstrate this, we produced a chart showing the theoretical execution times for running a file containing 25,600,000 reads with different degrees of parallelism (see Figure 25). The chart is an estimation of our performance scalability based on the measurements gathered in Figure 23. Here, we plot the total time for executing an `AlignJob`, the total time for the `align` stage and the ideal total time. These estimated times are plotted against the degree of parallelism, using a logarithmic scale for time to better highlight differences at high degrees of parallelism.

The "Total Time", "Align Time" and "Ideal Total Time" are based on the following formula:

$$t_{total} = \frac{t_{ar}.r}{p} + r.t_{pr} + t_c$$

Where t_{ar} is the time to align a single read, r is the number of reads in the reads file, p is the level of parallelism, t_{pr} is the time to process a single read in `sampad` and `aln2sam` and t_c is the constant time overhead for operations which do not scale with the number of reads (AKS overheads).

The “Total Time” is calculated as the total time to run the `align`, `merge`, `aln2sam` and `sampad` stages in addition to AKS overheads. The “Align Time” is calculated as the time to align a single read multplied by the total number of reads. This is then divided by the level of parallelism and the time taken to schedule the align `Job` is added to estimate how long alignment is expected to take. Finally, “Ideal Total Time” is calculated as the estimated Total Time divided by the level of parallelism.

This model aligns well with our observed performance characteristics and enables us to predict performance for much higher degrees of parallelism, as well as much larger read files. This in turn enables us to derive interesting conclusions about how best to leverage K8s-BWBBLE in these situations.

Initially we get a large return on investment relative to the degree of parallelism. We can reduce the time to complete and alignment from `01:32:43` to `00:15:56` by adding 6 additional `Pods`. Effectively, each pod reduces the time taken by about 12 minutes. However, to get below 10 minutes we need to add an additional 4 `Pods` with each additional pod only reducing the time taken by about 1.5 minutes. As we seek lower runtimes, we experience diminishing returns on adding additional compute resources.

Using the above model with a parallelism of 8 and the measured AKS overhead of 34 seconds (corresponding to a t_c of `00:01:42`), we see an end-to-end execution time of `00:08:31`. If we simulate an AKS overhead of 3 seconds instead (a t_c of `00:00:09`), the end-to-end execution time only drops to `00:06:57`. This highlights that the limiting factor in reducing the end-to-end time of BWBBLE is the `aln2sam` and `sampad` stages. If we can reduce the time spent in these stages, we can more closely approximate linear scaling for the entire workflow.

Due to the way in which `aln2sam` and `sampad` work, we can likely improve their execution times by focussing efforts on reducing time spent in I/O operations. This can be done by investing in faster I/O hardware (NVMe storage etc) as well as rewriting BWBBLE’s I/O components to leverage asynchronous APIs and a pipeline architecture. This architecture would enable BWBBLE to perform work while waiting on I/O operations to complete and could reduce the impact of I/O latency on throughput.

Other improvements would likely focus on how information is loaded into memory, minimizing allocations for each operation and potentially re-using loaded data across multiple jobs by converting these components into long-running services.

5.1 Performance Breakdown

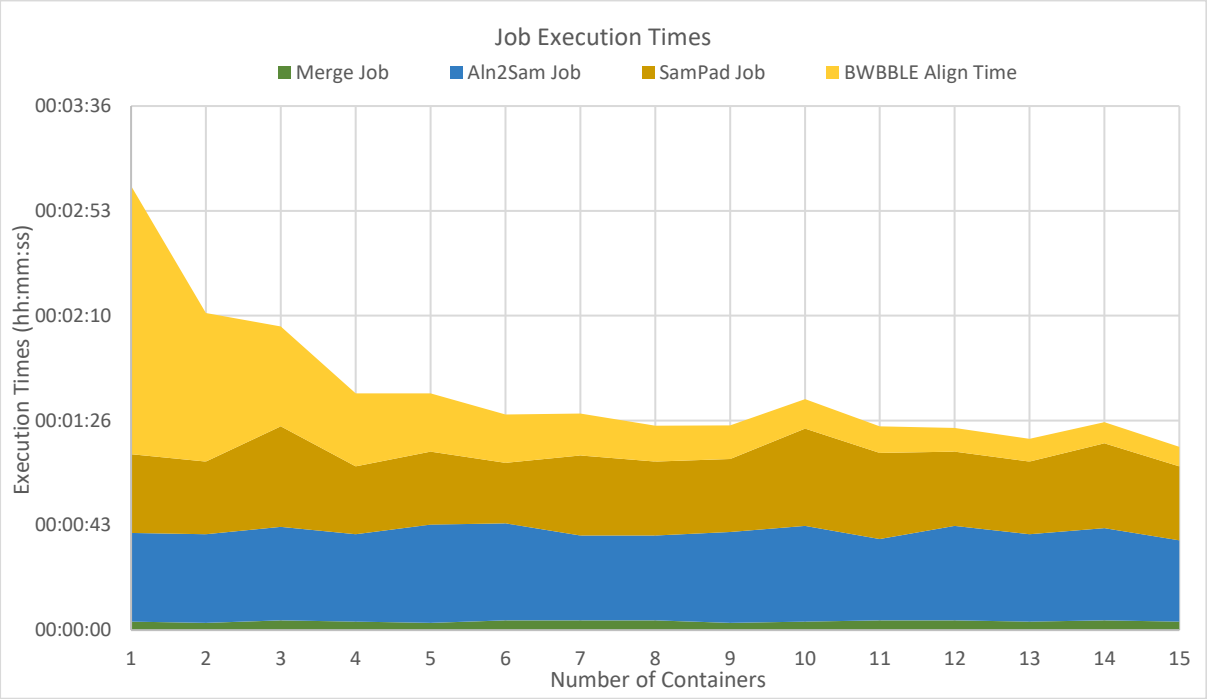

Figure 26 Total execution times for all BWBBLE Jobs

Figure 26 shows a stacked area chart which highlights the contributions that individual BWBBLE stages make to the total execution time. We can see the point at which we stop gaining the benefit of adding extra containers to the `align` stage and where we start to see diminishing returns.

This behaviour is caused by the fact that the `merge`, `aln2sam` and `sampad` jobs are all non-parallelizable. Due to this, they behave as a fixed cost overhead for the execution of an `AlignJob` regardless of the parallelism. While grouping these stages together into a single Kubernetes `Job` might reduce some of the associated overheads and reduce total execution time, in practice there is little we can do without parallelizing these other stages if we wish to see improvements with increased parallelism.

Figure 26 also clearly highlights how much faster the `merge` stage is relative to the other stages. In K8s-BWBBLE, `merge` is the only stage that does not use ACIs for running its workload, as well as the only stage which uses `busybox` instead of `ubuntu` as its base image. The `merge` stage uses standard Kubernetes worker nodes ("Standard_B2s" VMs with 4GB RAM and 2 vCPUs) to run the workload. To explore the implications of using Kubernetes worker nodes instead of ACIs, we re-ran a portion of the benchmark using worker nodes for the other stages.

This involved adding two more worker nodes to the cluster (bringing the total amount of resources to 6 vCPUs and 12GB of RAM) and changing the `use_aci` flag to `False` in the parameters we pass to create a job with the controller.

We ran `AlignJobs` with parallelisms of 1 to 8 in this configuration, avoiding higher parallelisms due to the limited resources available on our cluster of worker nodes. This highlighted one of the key reasons we chose to use ACIs for the project: that doing so enables higher levels of parallelism.

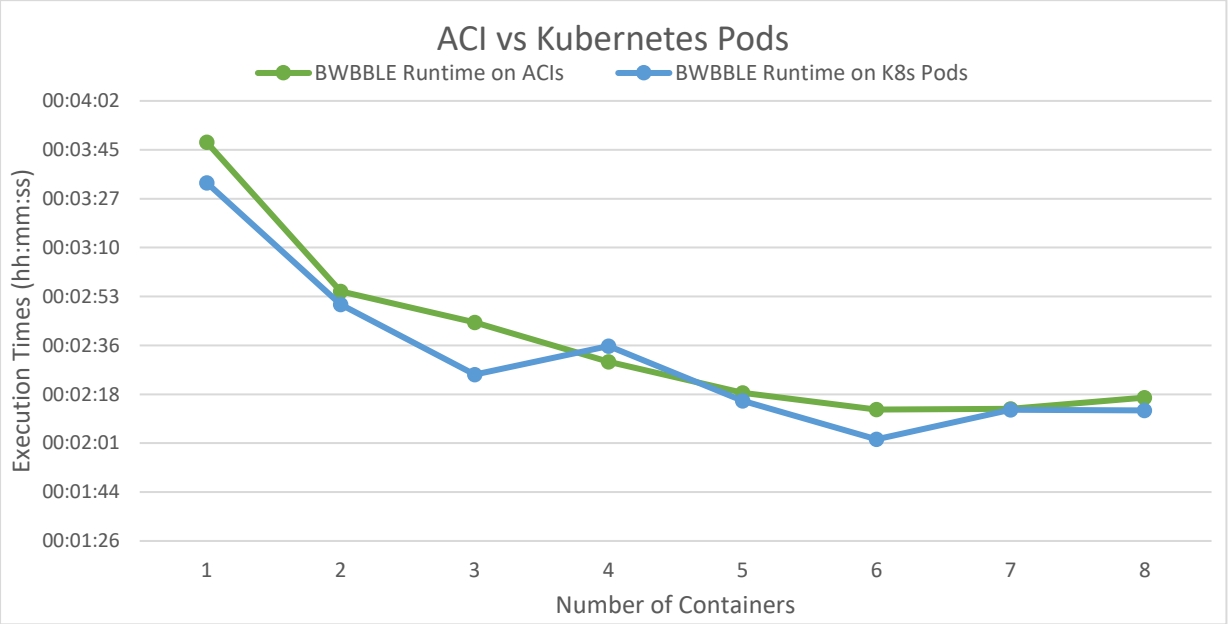

Figure 27 AlignJob runtime with ACIs vs without ACIs

The data revealed that K8s-BWBBLE completed `AlignJobs` slightly faster on standard Kubernetes nodes when compared to ACIs. This was expected as we have our own dedicated infrastructure and can cache the BWBBLE container images on these nodes. With our own dedicated infrastructure, scheduling becomes simpler with less probability of containers failing. We also no longer need to account for the latency in communicating with Azure.

However, we did expect to see a greater reduction in the runtime than we did when switching to standard Kubernetes nodes from ACIs. This revealed the possibility that there a shared bottleneck preventing us from achieving greater speedups.

There are a number of things which may account for this bottleneck, including the relatively large number of volume mounts which need to occur for each pod, the size of the container image being started, the need to copy data over the network etc.

5.2 Overheads

To further evaluate the overheads associated with running BWBBLE stages on Kubernetes, we extended the controller to gather information about the time spent within the BWBBLE `align` stage, as well as the time taken to complete the Kubernetes `Job`. This enabled us to plot Figure 28, which shows the time spent within BWBBLE's `align` stage as well as the overhead of running the program within a Kubernetes `Pod`.

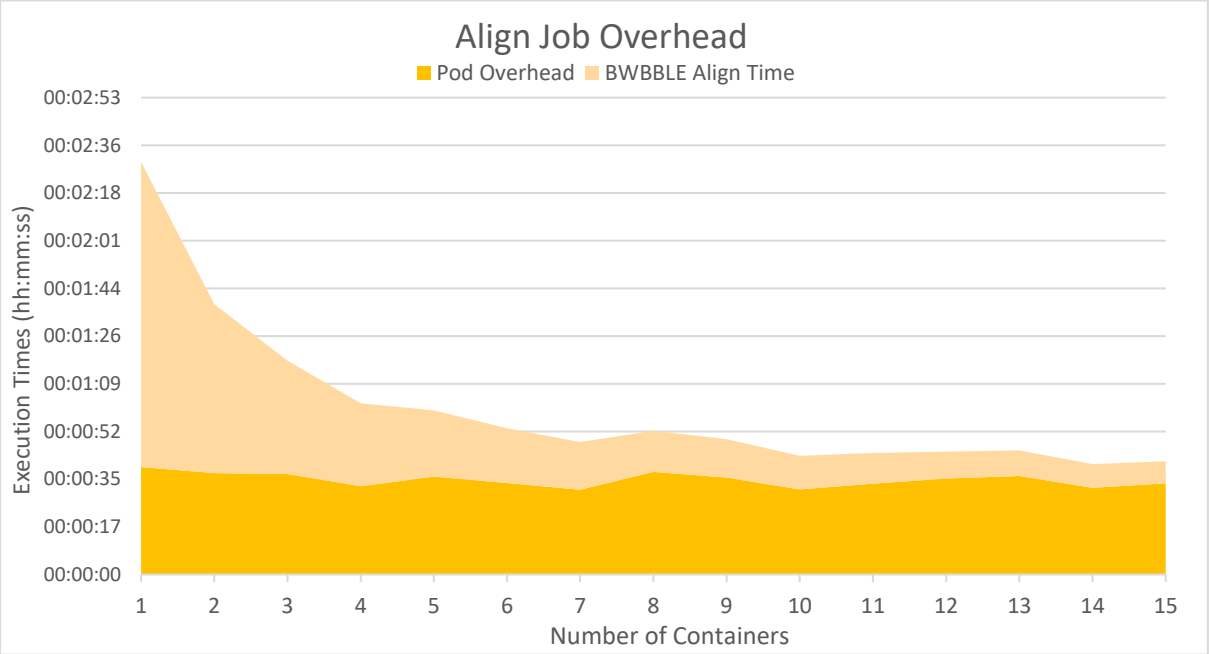

Figure 28 Execution time overhead for align stage

The `Pod` overhead is measured from when the `Pod` is first created until it finally completes execution. This covers time taken to start and schedule the `Pod`, as well as any restarts should the `Pod` fail to run successfully. By taking the median of these measurements, we have excluded situations in which `Pods` failed to run successfully.

This chart helps us understand the limitations of running BWBBLE on a Kubernetes cluster. Although we do see a reduction in execution times as we increase parallelism, we reach a point where we no longer experience the same benefits for each additional container. Although the time taken to complete the alignment does continue to decrease, we spend more time setting up the Pod than we do running the containerized program.

In theory, we could reduce this overhead by removing Kubernetes and Docker altogether, using our own dedicated VMs in much the same way as AWS-BWBBLE; however this is not aligned with the project's objective of making read alignment at scale more accessible. Another alternative would be

to switch out Kubernetes for Nomad, a more specialized workload scheduler than Kubernetes. This option is discussed in Chapter 7.

Another area of improvement involves reducing the size of the BWBBLE container image by switching to a smaller base image. Currently, we use Ubuntu as our base image which is 188MB in size. We could use the Alpine base image which is only 5MB to reduce the time taken to download the image from DockerHub and start the container.

To investigate whether the degree of parallelism impacted the Pod overhead in any way, we plotted the minimum and maximum overhead measurements for each parallelism. This is shown in Figure 29 and it highlights that in general there is not a significant increase in overhead as parallelism increases. It is interesting to note, however, that in situations where `Jobs` needed to be retried due to `Pod` failures, the maximum overhead is significantly higher than in other cases.

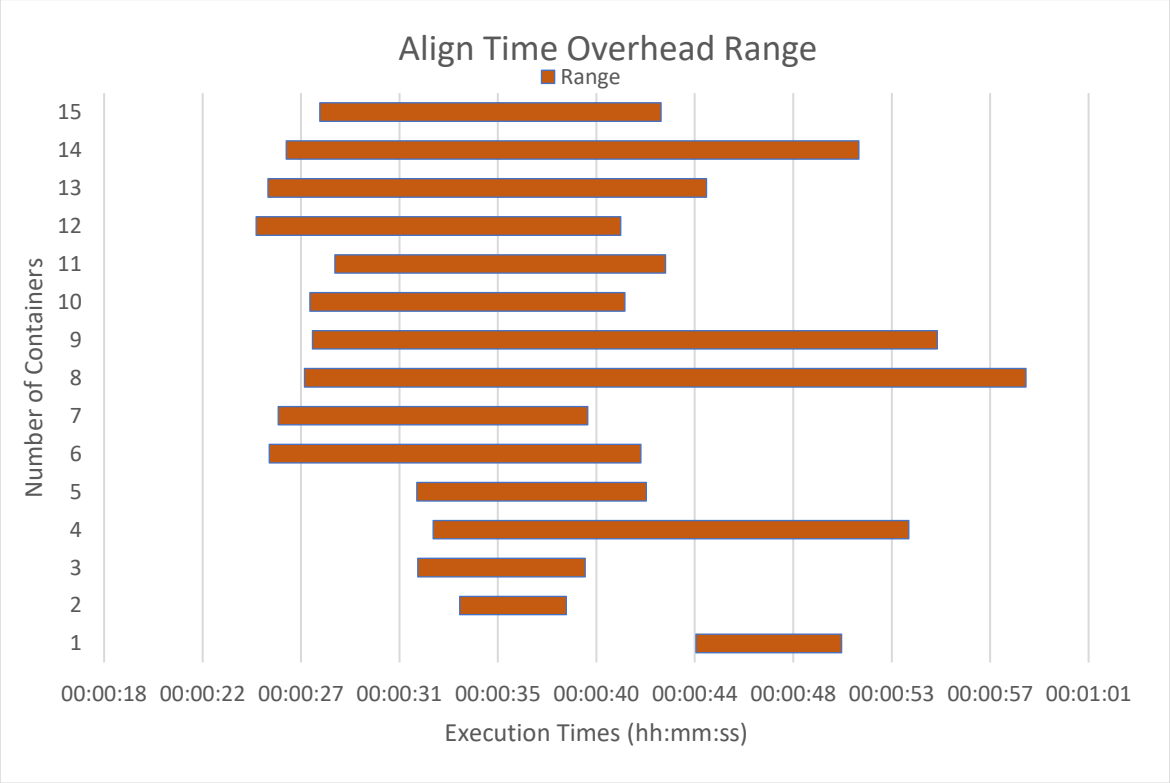

Figure 29 Range of execution time overheads for Align Job

It is also interesting to note that when comparing the time taken for the first `Job` to the last `Job` in a parallelized alignment, we see a uniform increase in execution time. This may be attributed to the way in which the `reads.fastq` file is processed – with later jobs needing to scan through more of the file to identify the portion they are responsible for aligning. This could be avoided by using `fseek` and passing fixed byte offsets to `mg-aligner`, rather than line numbers.

6 Conclusion

This project succeeded in reducing the time taken for geneticists to use BWBBLE to perform a DNA alignment, demonstrating the ability for a researcher to easily and cost effectively achieve a 91% scaling factor in the BWBBLE `align` stage. When comparing the end-to-end time taken to perform an alignment, the additional ease of use and simplified infrastructure management provided by Kubernetes and our controller offer clear advantages over prior solutions.

The question of whether this project offers the fastest way to perform an individual DNA alignment with BWBBLE is more complicated, however. With smaller DNA reads files, the overheads involved with using Kubernetes and ACIs mean that this approach may be slower than alternatives. This is due to the impact of Amdahl's law on partially parallelizable programs like BWBBLE. As such, without parallelizing other BWBBLE stages like `aln2sam` and `sampad`, we will be limited in terms of how quickly we can perform a minimal read alignment. On the other hand, when working with much larger read files, this approach can have significant demonstrable benefits for research time – reducing the total alignment time significantly.

This project has also demonstrated the advantages of using a declarative approach to workflow execution in the Cloud, when compared to an imperative approach such as that employed by AWS-BWBBLE. The primary benefits of this approach are a vastly simpler execution model, the ability to recover from failure effectively and the ability to easily parallelize individual stages with minimal additional effort. Kubernetes offers several low-level declarative constructs for this purpose and the ability to extend the Kubernetes API with high-level concepts like our `AlignJob` makes this feature set available for use by a broad audience.

By exposing this simplified, high-level API, we have made it easier for geneticists to clearly define and execute their workloads in a secure and scalable manner. In doing so, we meet the second objective for this project by reducing the time taken to perform this task.

Due to the added exposure of cloud-based workloads, it was necessary for us to carefully consider the security aspects of this project's design. Integration with Azure AD and careful application of Kubernetes RBAC controls helps us ensure that the sensitive genomic data processed by our cluster is kept secure.

Finally, by leveraging ACIs for costly processing work on the cluster, we have managed to keep the cost of running highly parallelized DNA alignment to a minimum, enabling researchers on a limited budget to rapidly analyse very large datasets. This allows us to satisfy the third objective of this project, which is to make cloud based read alignment more accessible.

7 Future Work

In this section we outline opportunities for improvement to reduce the runtime of BWBBLE whilst keeping the program accessible. The two main areas for potential improvements we see are in reducing the container start-up time and increasing application throughput.

Expanding upon the work in this project, it may be possible to parallelize the `aln2sam` and `sampad` stages by identifying ways to merge these output formats. If merging of a SAM file can be achieved, it would be possible to execute parallelized `aln2sam` and `sampad` operations – offering the potential for further reductions in overall execution time and magnifying any other improvements made to application throughput and container start-up time.

To boost application throughput, future efforts should invest in moving BWBBLE's I/O onto a non-blocking I/O framework. Doing so will enable various BWBBLE stages to perform work while waiting for new data to become available. The benefit of doing so will be most noticeably felt in environments (like Azure) where storage latency is high.

Another area of improvement involves reducing the amount of data held in memory before being written to disk. In some BWBBLE stages, the entire input is read into memory before processing begins, resulting in a significant number of unnecessary allocations in situations where the data does not need to be used again. Instead, this data can be read, processed and written to disk one record at a time. In conjunction with a non-blocking I/O implementation, this could drastically reduce the amount of time spent waiting for memory and storage. One such place this refactoring could happen is the `compute_C` method in `mg-aligner/bwt.c`.

There are also situations in which BWBBLE loads a large amount of data to perform lookups. This data often remains constant across multiple executions of the program and if BWBBLE were rewritten to act as a long-running service; the overhead of loading this data into memory for each job could be reduced significantly. In doing so, BWBBLE itself would become more like the controller we have created – with the benefit that container and application start-up time would be eliminated from the time taken to run an alignment.

To reduce the container start-up time and staying with the current K8s-BWBBLE architecture, we can also combine the multiple stages in BWBBLE into a single pod. This would mean we only need to schedule a pod once which would contain the logic to run all the BWBBLE stages. The container start-up time could also be reduced slightly by scheduling the containers onto dedicated hardware; however, this increases cost significantly.

In hindsight, Kubernetes worked well for this project. It greatly simplified running highly parallelizable jobs with its scheduling abilities. However, our benchmarking showed that Kubernetes does have an 8% overhead associated with it in the context of running a BWBBLE read alignment. The I/O, scheduling, networked storage, pulling down the container image and setting up the container contribute to the total execution time of completing a full alignment. To reduce this overhead, Kubernetes could be replaced with Nomad. Nomad is another orchestrator which has an optimistically concurrent scheduler. This makes it much faster than Kubernetes especially for larger workloads. Nomad is also capable of orchestrating applications directly, removing the need to use containers and potentially reducing the start-up time slightly.

From a user experience perspective, improvements could be made through the development of an external application which enables uploading of relevant files to the `input` Azure File Share, downloading of results from the `output` file share and the creation of `AlignJobs` through the Kubernetes API. By building a dedicated application, it would be possible to further reduce the complexity of operating this service and increase accessibility.

References

Abuín, J. M., Pichel, J. C., Pena, T. F. & Amigo, J., 2015. BigBWA: approaching the Burrows–Wheeler aligner to Big Data technologies. *Bioinformatics,* 15 December, 31(24), pp. 4003-4005.

Abuín, J. M., Pichel, J. C., Pena, T. F. & Amigo, J., 2016. SparkBWA: Speeding Up the Alignment of High-Throughput DNA Sequencing Data. *PLOS One,* 16 May, 11(5), pp. 1-21.

Arram, J., Kaplan, T., Luk, W. & Jiang, P., 2017. Leveraging FPGAs for Accelerating Short Read Alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatic,* May.14(3).

Boettiger, C., 2014. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts.,* Thu 10.pp. 71-79.

Buckingham, K. J. et al., 2009. Exome sequencing identifies the cause of a Mendelian disorder. *Nature Genetics,* November.

Chen, Y.-T.et al., 2015. *CS-BWAMEM: A fast and scalable read aligner at the.* Dublin, Ireland, High Throughput Sequencing Algorithms and Applications (HITSEQ) Poster Session.

Durbin, M., 2009. *So Long, Data Depression.* [Online] Available at: http://www.genomeweb.com/informatics/so-long-data-depression

Erlich, Y. & Narayanan, A., 2014. Routes for breaching and protecting genetic privacy. *Nature Review Genetics,* 08 May, Volume 15, p. 409–421.

Genomics Education Programme, n.d. *Genomics Education Programme Image Library.* [Online] Available at: https://farm8.staticflickr.com/7429/13081113544_d05a3d0b2b_z.jpg

Huang, L., 2015. *bwbble mg-ref.* [Online] Available at: https://github.com/viq854/bwbble/tree/master/mg-ref

Huang, L., Popic, V. & Batzoglou, S., 2013. Short read alignment with populations of genomes. *Bioinformatics,* July, 29(13), p. i361–i370.

Keane, T. M. et al., 2011. Mouse genomic variation and its effect on phenotypes and gene regulation. *Nature,* Sep.Volume 477.

Kumar, S., Tamura, K. & Nei, M., 2004. MEGA3: Integrated software for Molecular Evolutionary Genetics Analysis and sequence alignment.. *Briefings in Bioinformatics.*

Lambert, C., Fernandes, M., Decouchant, J. e. & Esteves-Verissimo, P., 2018. *MaskAl: Privacy Preserving Masked Reads.* Salvador, Brazil, s.n.

Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. L., 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology,* Volume 10.

Li, H. & Durbin, R., 2009. Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics,* 25(14), pp. 1754-60.

Li, H. & Homer, N., 2010. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics,* 11(5).

Li, R. et al., 2009. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics,* 25(15).

Malin, B., 2002. *Compromising Privacy with Trail Re-Identification : The REIDIT Algorithms,* s.l.: Carnegie Mellon University. Center for Automated Learning and Discovery.

Malin, B. & Sweeney, L., 2004. "How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems.. *Journal of biomedical informatic,* 37(3), p. 179–192.

McGinley, K., 2019. *Parallel DNA Read Alignment Using the Amazon Cloud,* Dublin: Trinity College Dublin.

Pedrosa, L. et al., n.d. *Large-scale cluster management at Google with Borg.* s.l., s.n.

Peng, C. et al., 2012. *Vdn: Virtual machine image distribution network for cloud data centers.* s.l., IEE, p. 181–189.

Popic, V. & Batzoglou, S., 2017. A hybrid cloud read aligner based on MinHash and kmer voting that preserves privacy. *Nature Communications,* March.

Roach, J. et al., 2010. Analysis of Genetic Inheritance in a Family Quartet by Whole Genome Sequencing. *Science,* p. 636–639.

Schneeberger:, K. et al., 2009. Simultaneous alignment of short reads against multiple genomes. *Genome Biology,* 17 Sep.10(9).

Stein, L. D., 2010. The case for cloud computing in genome informatics. *Genome Biology,* 11(5).

Stratford, B., 2018. *Cloud-based high speed parallel DNA read alignment using the Burrows-Wheeler Transform,* Dublin: Trinty College Dublin.

Trapnell, C. & Salzberg, S. L., 2009. How to map billions of short reads onto genomes. *Nature biotechnology,* 27(5), pp. 455-7.

Xiao, Z., Song, W. & Chen, Q., 2013. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems,,* June, 24(6), p. 1107–1117.

Zhang, Q. et al., 2018. *A Comparative Study of Containers and Virtual Machines in Big Data Environment.* s.l., s.n.

Zhang, Q., Liu, L. & Ren, J., 2016. *iballoon: Efficient vm memory balancing as a service.* s.l., IEE, pp. 33-40.

Appendices

Appendix 1 mg-ref/Dockerfile

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y gcc g++ make zlib1g-dev

ADD . /src
WORKDIR /src
RUN make all

FROM ubuntu:latest
COPY --from=0 /src/comb /usr/bin/comb
COPY --from=0 /src/data_prep /usr/bin/data_prep
COPY --from=0 /src/sam_pad /usr/bin/sam_pad
ADD ./entrypoint.sh /usr/bin/entrypoint
ADD ./help.sh /usr/bin/help
RUN chmod +x /usr/bin/entrypoint /usr/bin/help

RUN apt-get update && apt-get install -y libgomp1

WORKDIR /
ENTRYPOINT [ "/usr/bin/entrypoint" ]
CMD ["/usr/bin/help"]
```

Appendix 2 mg-aligner/Dockerfile

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y gcc g++ make zlib1g-dev

ADD . /src
WORKDIR /src
RUN make all

FROM ubuntu:latest
COPY --from=0 /src/bwbble /usr/bin/bwbble
ADD entrypoint.sh /usr/bin/entrypoint

RUN chmod +x /usr/bin/entrypoint
RUN apt-get update && apt-get install -y libgomp1

WORKDIR /
ENTRYPOINT ["/usr/bin/entrypoint"]
CMD []
```

Appendix 3 mg-ref/entrypoint.sh

```
#!/usr/bin/env bash

set -e
set -o pipefail

# Set the application to the $APPLICATION environment variable, or the 1st argument
to the script if that isn't set.
APPLICATION=${APPLICATION-$1}
# If neither of the above is set, then use data_prep
APPLICATION=${APPLICATION-data_prep}

ARGS_FILE="${ARGS_FILE-/var/run/container_args}"

# if the command line arguments file exists, then use that
if [[ -f $ARGS_FILE ]]; then
    cat $ARGS_FILE | xargs /usr/bin/$APPLICATION
else
    exec /usr/bin/$APPLICATION ${@:2}
fi
```

Appendix 4 mg-aligner/entrypoint.sh

```
#!/usr/bin/env bash
set -e
set -o pipefail

ARGS_FILE="${ARGS_FILE-/var/run/container_args}"

# if the command line arguments file exists, then use that
if [[ -f $ARGS_FILE ]]; then
    cat $ARGS_FILE | xargs /usr/bin/bwbble
else
    exec /usr/bin/bwbble $@
fi
```

Appendix 5 Kubernetes/bwbble/values.yaml

```yaml
# Default values for mg-align-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.
replicaCount: 1
dataprep:
  enabled: true
  image:
    repository: bwbble/mg-ref
    tag: 250
  ref_genome: chr21.fasta
  snp_indel_files:
    - ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf
  # These files are the output of the data preparation phase
  snp_file: ref_w_snp.fasta
  bubble_file: bubble.data
  snp_bubble_file: ref_w_snp_and_bubble.fasta
index:
  enabled: true
align:
  enabled: true
  image:
    repository: bwbble/mg-aligner
    tag: latest
  # These files are input for the align phase, generated by the dataprep phase
  snp_file: ref_w_snp.fasta
  bubble_file: bubble.data
  output: output.sam
  reads:
    file: sim_chr21_N100.fastq
    ranges:
      - start: 0
        length: 50
      - start: 50
        length: 50
ref:
  image:
    repository: bwbble/mg-ref
    tag: 250
  vcf:
    file: ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf
  human_genome:
    file: human_g1k_v37.fasta
image:
  pullPolicy: IfNotPresent
nameOverride: ""
fullnameOverride: ""
storageSecrets:
  username: Guest
  password: ""
volumes:
  input: {}
  refoutput: {}
  alignoutput: {}
```

Appendix 6 mg-aligner support for read file skips (diff).

```
diff --git a/mg-aligner/align.c b/mg-aligner/align.c
index fb4cdf1..faa4f57 100644
--- a/mg-aligner/align.c
+++ b/mg-aligner/align.c
@@ -35,6 +35,8 @@ void set_default_aln_params(aln_params_t* params) {
    params->max_best = 30;
    params->no_indel_length = 5;
    params->n_threads = 1;
+   params->lines_skip = 0;
+   params->lines_limit = -1;
 }

 int align_reads(char* fastaFname, char* readsFname, char* alnsFname, aln_params_t* params)
 {
@@ -
52,7 +54,7 @@ int align_reads(char* fastaFname, char* readsFname, char* alnsFname, aln_para
ms_
    printf("Total BWT loading time: %.2f sec\n", (float)(clock() - t) / CLOCKS_PER_SEC);

    t = clock();
-   reads_t* reads = fastq2reads(readsFname);
+   reads_t* reads = fastq2reads(readsFname, params->lines_skip, params->lines_limit);
    printf("Total read loading time: %.2f sec\n", (float)(clock() - t) / CLOCKS_PER_SEC);


    sa_intv_list_t* sa_intv_table = NULL;
@@ -
506,7 +508,7 @@ void alns2sam(char *fastaFname, char *readsFname, char *alnsFname, char* sa
mFnam
    int num_alns;
    alns_t* alns = alnsf2alns_bin(&num_alns, alnsFname);
    //alns_t* alns = alnsf2alns(&num_alns, alnsFname);
-   reads_t* reads = fastq2reads(readsFname);
+   reads_t* reads = fastq2reads(readsFname, 0, -1);
    //assert(num_alns == reads->count);

    // open SAM for writing
@@ -
672,7 +674,7 @@ void eval_alns(char *fastaFname, char *readsFname, char *alnsFname, int is_
multi
    int num_alns;
    alns_t* alns = alnsf2alns(&num_alns, alnsFname);
    bwt_t* BWT = load_bwt(bwtFname, 1);
-   reads_t* reads = fastq2reads(readsFname);
+   reads_t* reads = fastq2reads(readsFname, 0, -1);
    assert(num_alns == reads->count);

    // for each read: evaluate the alignment quality and accuracy
diff --git a/mg-aligner/align.h b/mg-aligner/align.h
index a55aea0..18376e0 100644
--- a/mg-aligner/align.h
+++ b/mg-aligner/align.h
@@ -76,6 +76,10 @@ typedef struct {
    // multi-threading
    int n_threads;
```

```
+   // the lines to start and end reading within the reads file
+   int lines_skip;
+   int lines_limit;
+
 } aln_params_t;

 typedef struct {
diff --git a/mg-aligner/io.c b/mg-aligner/io.c
index bc8c2c4..4e3cb46 100644
--- a/mg-aligner/io.c
+++ b/mg-aligner/io.c
@@ -
407,7 +407,7 @@ void seq2rev_compl(unsigned char* seq, bwtint_t seqLen, unsigned char** rcS
eq) {
 /* Reads I/O */

 // loads the read sequences from the FASTQ file
-reads_t* fastq2reads(const char *readsFname) {
+reads_t* fastq2reads(const char *readsFname, int skip, int limit) {
     FILE *readsFile = (FILE*) fopen(readsFname, "r");
     if (readsFile == NULL) {
         printf("load_reads_fastq: Cannot open reads file: %s !\n", readsFname);
@@ -418,6 +418,14 @@ reads_t* fastq2reads(const char *readsFname) {
     reads->reads = (read_t*) malloc(allocatedReads*sizeof(read_t));
     reads->count = 0;

+   skip *= 4;
+   while (skip-- > 0 && !feof(readsFile)) {
+       while (!feof(readsFile) && '\n' != (char) getc(readsFile)) {
+           // Do nothing with this character, it is being skipped while we look
+           // for a new-line.
+       }
+   }
+
     char c;
     while(!feof(readsFile)) {
         if (reads->count >= allocatedReads) {
@@ -507,6 +515,10 @@ reads_t* fastq2reads(const char *readsFname) {
             reads->max_len = read->len;
         }
         reads->count++;
+
+       if (reads->count == limit) {
+           break;
+       }
     }
     printf("Loaded %d reads from %s.\n", reads->count, readsFname);

diff --git a/mg-aligner/io.h b/mg-aligner/io.h
index 3d42777..f20867d 100644
--- a/mg-aligner/io.h
+++ b/mg-aligner/io.h
@@ -
210,7 +210,7 @@ void fasta2ref(const char *fastaFname, const char* refFname, const char* an
nFnam
 void ref2seq(const char* refFname, unsigned char** seq, bwtint_t* seqLen);
 void fasta2pac(const char *fastaFname, const char* pacFname, const char* annFname);
 void pac2seq(const char *pacFname, unsigned char** seq, bwtint_t *seqLength);
```

```
-reads_t* fastq2reads(const char *readsFname);
+reads_t* fastq2reads(const char *readsFname, int skip, int limit);
 void seq2rev_compl(unsigned char* seq, bwtint_t seqLen, unsigned char** rcSeq);
 void parse_read_mapping(read_t* read);

diff --git a/mg-aligner/main.c b/mg-aligner/main.c
index 9f31546..e88e74e 100644
--- a/mg-aligner/main.c
+++ b/mg-aligner/main.c
@@ -63,6 +63,8 @@ static int align_usage() {
    printf("         o    maximum number of gap opens (default: 1)\n");
    printf("         e    maximum number of gap extends (default: 6) \n");
    printf("         t    run multi-threaded with t threads (default: 1)\n");
+   printf("         s    the number of lines to skip in the read file before processing (d
efault: 0)\n");
+   printf("         p    the number of lines to process in the read file (default: -
1)\n");
    printf("         S    align with a single-genome reference\n");
    printf("         P    use pre-calculated partial alignment results\n");
    printf("\n");
@@ -97,7 +99,7 @@ int main(int argc, char *argv[]) {
        set_default_aln_params(params);

        int c;
-       while ((c = getopt(argc-1, argv+1, "M:O:E:n:k:o:e:l:m:t:SP")) >= 0) {
+       while ((c = getopt(argc-1, argv+1, "M:O:E:n:k:o:e:l:m:t:s:p:SP")) >= 0) {
                switch (c) {
                    case 'M': params->mm_score = atoi(optarg); break;
                    case 'O': params->gapo_score = atoi(optarg); break;
@@ -109,6 +111,8 @@ int main(int argc, char *argv[]) {
                    case 'l': params->seed_length = atoi(optarg); break;
                    case 'm': params->max_entries = atoi(optarg); break;
                    case 't': params->n_threads = atoi(optarg); break;
+                   case 's': params->lines_skip = atoi(optarg); break;
+                   case 'p': params->lines_limit = atoi(optarg); break;
                    case 'S': params->is_multiref = 0; break;
                    case 'P': params->use_precalc = 1; break;
                    case '?': align_usage(); return 1;
```

Appendix 7 Kubernetes/bwbble/templates/merge.yaml

```
{{ if .Values.align.enabled }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-merge-{{ .Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: merge
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: merge
      labels:
        app.kubernetes.io/managed-by: faaideen
    spec:
      restartPolicy: Never
      initContainers:
      - name: wait-for-align-job
        image: bitnami/kubectl
        args:
        - "wait"
        - "--for=condition=complete"
        - "job"
        - "-l"
        - "bwbble-job={{ .Release.Name }}"
        - "-l"
        - "bwbble-stage=align"
        - "--timeout=-1s"
      containers:
        - name: merge
          image: busybox
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          args:
            - "sh"
            - "-c"
            - "cat {{ range .Values.align.reads.ranges }} /mg-align-
output/{{ $.Release.Name }}.aligned_reads.{{ .start }}.aln {{ end }} > /mg-align-
output/{{ .Release.Name }}.aligned_reads.aln"
          volumeMounts:
          - mountPath: /input
            name: input
          - mountPath: /mg-ref-output
            name: ref-output
          - mountPath: /mg-align-output
            name: align-output
      volumes:
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 10 }}
{{ end }}
```

Appendix 8 Kubernetes/bwbble/templates/aks.values.yaml

```yaml
dataprep:
  enabled: false
  ref_genome: chr21.fa
  snp_indel_files:
    - ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf
  # These files are the output of the data preparation phase
  # When dataprep is enabled use ref_w_snp.fasta for snp_file
  snp_file: chr21_ref_w_snp.fasta
  bubble_file: chr21_bubble.data
  snp_bubble_file: chr21_ref_w_snp_and_bubble.fasta

index:
  enabled: true

align:
  enabled: true

  # These files are input for the align phase, generated by the dataprep phase
  # When dataprep is enabled use ref_w_snp.fasta for snp_file
  snp_file: chr21_ref_w_snp.fasta
  bubble_file: chr21_bubble.data
  output: output.sam
  reads:
    file: dummy_reads.fastq
    ranges:
      - start: 0
        length: -1
      # - start: 128000
      #   length: 128000
      # - start: 256000
      #   length: 128000

ref:
  vcf:
    file: ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf
  human_genome:
    file: chr21.fa

volumes:
  input:
    azureFile:
      secretName: azure-secret
      shareName: input
      readonly: false
  refoutput:
    azureFile:
      secretName: azure-secret
      shareName: ref-output
      readonly: false
  alignoutput:
    azureFile:
      secretName: azure-secret
      shareName: align-output
      readonly: false
```

Appendix 9 Scripts/controller.py

```
#!/usr/bin/env python
import hashlib
import string
import random
import logging
import yaml
from pprint import pprint
import sys
import os
import time
from typing import List
import subprocess
from kubernetes import client, config, utils
import kubernetes.client
from kubernetes.client.rest import ApiException, BatchV1Api, ApiClient, Configuration
import re
from datetime import timedelta
from math import floor

class Range(object):
    def __init__(self, start: int, length: int = -1):
        self.start = start
        self.length = length

    @property
    def name(self) -> str:
        if self.length == -1:
            return f"{self.start}-end"
        return f"{self.start}-{self.start+self.length}"

    @staticmethod
    def generate(num_reads: int, parallelism: int = 1):
        rrange = floor(num_reads / parallelism)
        ranges = []
        for i in range(0, parallelism):
            ranges.append(Range(i * rrange, rrange))

        ranges[len(ranges) - 1].length = -1

        return ranges

# Configuration
bwbble_container_image_version = "313"
reads_file = "dummy_reads_large.fastq"
bubble_file = "chr21_bubble.data"
snp_file = "chr21_ref_w_snp_and_bubble.fasta"
parallelism = 1
reads = 512000
file_ranges = Range.generate(reads, parallelism)

# Setup logging
logging.basicConfig(stream=sys.stdout, level=logging.INFO)

# Setup K8 configs

config.load_kube_config()
```

```
configuration = Configuration()
api_client = BatchV1Api(ApiClient(configuration))

api_instance = kubernetes.client.BatchV1Api(api_client)

def execution_times(namespace: str, release: str, stage: str, align_logs: bool = False):
    # config.load_kube_config()
    # pod_name = "bwbble-align-dummylargereads1-range-0--1-799ms"
    try:
        if align_logs:
            # get execution time for pods
            api_response = kubernetes.client.CoreV1Api(api_client).list_namespaced_pod(
                namespace=namespace,
                label_selector=f"bwbble-release={release},bwbble-stage={stage}",
            )
            for item in api_response.items:
                logs = kubernetes.client.CoreV1Api(api_client).read_namespaced_pod_log(
                    item.metadata.name, namespace, container="align", tail_lines=100
                )
                with open(f"{namespace}-{item.metadata.name}.log", "w+") as f:
                    if logs:
                        f.write(logs)
                        f.close()
                        rem = re.search(
                            r"read alignment time: (\d+\.?\d*) sec", logs, re.IGNORECASE
                        )

                        if rem:
                            print(
                                item.metadata.name,
                                " (logs): ",
                                timedelta(seconds=float(rem[1])),
                            )
        # get execution time for pods
        api_response = api_instance.list_namespaced_job(
            namespace=namespace,
            label_selector=f"bwbble-release={release},bwbble-stage={stage}",
        )
        for item in api_response.items:
            if item.status.completion_time:
                print(
                    item.metadata.name,
                    " (job): ",
                    item.status.completion_time - item.status.start_time,
                )
            else:
                print(item.metadata.name, " (job): Not yet finished")

    except ApiException as e:
        print(e)

def create_job_resources(
    namespace: str,
    release: str,
    stage: str,
    container_image: str,
    args: List[str],
    use_config_map_args: bool = True,
```

```
    resources: client.V1ResourceRequirements = None,
    env: List[client.V1EnvVar] = None,
    name_suffix: str = "",
    use_aci: bool = True,
):
    labels = {
        "app.kubernetes.io/managed-by": "faaideen",
        "bwbble-release": release,
        "bwbble-stage": stage,
    }

    resources = resources or client.V1ResourceRequirements(
        limits={"memory": "2Gi", "cpu": "1"}, requests={"memory": "2Gi", "cpu": "1"}
    )
    created_resources = []
    env_array = []

    if use_aci and not use_config_map_args:
        raise Exception(
            "Azure Container Instances require that arguments are passed using a file"
        )

    if use_config_map_args:
        env_array.append(
            client.V1EnvVar(name="ARGS_FILE", value="/var/run/args/container_args")
        )

        created_resources.append(
            kubernetes.client.CoreV1Api(api_client).create_namespaced_config_map(
                namespace,
                kubernetes.client.V1ConfigMap(
                    api_version="v1",
                    kind="ConfigMap",
                    metadata=client.V1ObjectMeta(
                        name=f"bwbble-{release}-{stage}{name_suffix}", labels=labels
                    ),
                    data={
                        "container_args": " ".join(
                            [f"'{a}'" if " " in a else a for a in args]
                        ),
                    },
                ),
            )
        )

    if env is not None:
        env_array.extend(env)

    job_spec = client.V1Job(
        api_version="batch/v1",
        kind="Job",
        metadata=client.V1ObjectMeta(
            name=f"bwbble-{release}-{stage}{name_suffix}", labels=labels
        ),
        spec=client.V1JobSpec(
            template=client.V1PodTemplateSpec(
                metadata=client.V1ObjectMeta(name=stage, labels=labels),
                spec=client.V1PodSpec(
```

```
                restart_policy="Never",
                containers=[
                    client.V1Container(
                        name=stage,
                        image=container_image,
                        image_pull_policy="IfNotPresent",
                        args=args,
                        env=env_array,
                        resources=resources,
                        volume_mounts=[
                            client.V1VolumeMount(mount_path="/input", name="input"),
                            client.V1VolumeMount(
                                mount_path="/mg-ref-output", name="ref-output"
                            ),
                            client.V1VolumeMount(
                                mount_path="/mg-align-output", name="align-output"
                            ),
                        ],
                    )
                ],
                volumes=[
                    client.V1Volume(
                        name="input",
                        azure_file=client.V1AzureFileVolumeSource(
                            secret_name="azure-secret",
                            share_name="input",
                            read_only=True,
                        ),
                    ),
                    client.V1Volume(
                        name="ref-output",
                        azure_file=client.V1AzureFileVolumeSource(
                            secret_name="azure-secret",
                            share_name="ref-output",
                            read_only=False,
                        ),
                    ),
                    client.V1Volume(
                        name="align-output",
                        azure_file=client.V1AzureFileVolumeSource(
                            secret_name="azure-secret",
                            share_name="align-output",
                            read_only=False,
                        ),
                    ),
                ],
            ),
        )
    ),
)

if use_aci:
    job_spec.spec.template.spec.node_selector = {
        "kubernetes.io/role": "agent",
        "beta.kubernetes.io/os": "linux",
        "type": "virtual-kubelet",
    }
```

```
        job_spec.spec.template.spec.tolerations = [
            client.V1Toleration(key="virtual-kubelet.io/provider", operator="Exists"),
            client.V1Toleration(key="azure.com/aci", effect="NoSchedule"),
        ]

    if use_config_map_args:

        job_spec.spec.template.spec.volumes.append(
            client.V1Volume(
                name="args",
                config_map=client.V1ConfigMapVolumeSource(
                    name=f"bwbble-{release}-{stage}{name_suffix}"
                ),
            )
        )

        job_spec.spec.template.spec.containers[0].args = []
        job_spec.spec.template.spec.containers[0].volume_mounts.append(
            client.V1VolumeMount(mount_path="/var/run/args", name="args")
        )

    created_resources.append(
        BatchV1Api(api_client).create_namespaced_job(
            namespace, job_spec
        )
    )

    return created_resources

def wait_for_all_jobs(
    namespace: str, release: str, stage: str, resources: List[kubernetes.client.V1Job]
):
    watcher = kubernetes.watch.Watch()

    pending_jobs = set([r.metadata.name for r in resources])

    for event in watcher.stream(
        kubernetes.client.BatchV1Api(api_client).list_namespaced_job,
        namespace,
        label_selector=f"bwbble-release={release},bwbble-stage={stage}",
    ):

        if event["object"].status.completion_time:
            pending_jobs.remove(event["object"].metadata.name)

            if len(pending_jobs) == 0:
                watcher.stop()
                return

def run_data_prep(namespace: str, release: str):
    # Do the dataprep job
    api_responses = create_job_resources(
        namespace,
        release,
        "data-prep",
        f"bwbble/mg-ref:{bwbble_container_image_version}",
    )
```

```
    # Wait for the data-prep job to complete
    wait_for_all_jobs(
        namespace,
        release,
        "data-prep",
        [r for r in api_responses if isinstance(r, kubernetes.client.V1Job)],
    )
    print("**** All Jobs completed for dataprep phase of mg-ref ****")

    # Do the combine job
    api_responses = create_job_resources(
        namespace, release, "comb", f"bwbble/mg-ref:{bwbble_container_image_version}"
    )

    # Wait for the data-prep job to complete
    wait_for_all_jobs(
        namespace, [r for r in api_responses if isinstance(r, kubernetes.client.V1Job)]
    )
    print("**** All Jobs completed for comb phase of mg-ref ****")

    return api_responses

def run_index(namespace: str, release: str):
    api_responses = create_job_resources(
        namespace,
        release,
        "index",
        f"bwbble/mg-aligner:{bwbble_container_image_version}",
        args=["index", f"/mg-ref-output/{snp_file}"],
        resources=client.V1ResourceRequirements(
            limits={"memory": "2Gi", "cpu": "1"}, requests={"memory": "2Gi", "cpu": "1"}
        ),
    )

    # Wait for the index job to complete
    wait_for_all_jobs(
        namespace,
        release,
        "index",
        [r for r in api_responses if isinstance(r, kubernetes.client.V1Job)],
    )
    print("**** All Jobs completed for index phase of mg-aligner ****")

def run_align(namespace: str, release: str):
    alignment_jobs = []

    for range in file_ranges:
        api_responses = create_job_resources(
            namespace,
            release,
            "align",
            f"bwbble/mg-aligner:{bwbble_container_image_version}",
            args=[
                "align",
                "-s",
                f"{range.start}",
                "-p",
                f"{range.length}",
```

```
                f"/mg-ref-output/{snp_file}",
                f"/input/{reads_file}",
                f"/mg-align-output/{release}.aligned_reads.{range.name}.aln",
            ],
            name_suffix=f"-{range.name}",
        )

        alignment_jobs.extend(
            [r for r in api_responses if isinstance(r, kubernetes.client.V1Job)]
        )

    # Wait for the align job to complete
    wait_for_all_jobs(namespace, release, "align", alignment_jobs)
    print("All Jobs completed for align phase of mg-aligner")

    print("**** Starting the merge job ****")
    run_merge(namespace, release)

    print("**** Starting the aln2sam job ****")
    run_aln2sam(namespace, release)

    print("**** Starting the sam_pad job ****")
    run_sam_pad(namespace, release)

def run_merge(namespace: str, release: str):
    merge_command = [
        "cat",
        *[
            f"/mg-align-output/{release}.aligned_reads.{range.name}.aln"
            for range in file_ranges
        ],
        ">",
        f"/mg-align-output/{release}.aligned_reads.aln",
    ]

    api_responses = create_job_resources(
        namespace,
        release,
        "merge",
        "busybox:latest",
        use_config_map_args=False,
        use_aci=False,
        args=[
            "sh",
            "-c",
            " ".join([f"'{p}'" if " " in p else p for p in merge_command]),
        ],
        resources=client.V1ResourceRequirements(
            limits={"memory": "128Mi", "cpu": "1"},
            requests={"memory": "128Mi", "cpu": "50m"},
        ),
    )

    # Wait for the merge job to complete
    wait_for_all_jobs(namespace, release, "merge", api_responses)
    print("All Jobs completed for merge phase")

def run_aln2sam(namespace: str, release: str):
```

```
    api_responses = create_job_resources(
        namespace,
        release,
        "aln2sam",
        f"bwbble/mg-aligner:{bwbble_container_image_version}",
        args=[
            "aln2sam",
            f"/mg-ref-output/{snp_file }",
            f"/input/{reads_file}",
            f"/mg-align-output/{release}.aligned_reads.aln",
            f"/mg-align-output/{release}.aligned_reads.sam",
        ],
    )

    # Wait for the aln2sam job to complete
    wait_for_all_jobs(
        namespace,
        release,
        "aln2sam",
        [r for r in api_responses if isinstance(r, kubernetes.client.V1Job)],
    )
    print("**** All Jobs completed for aln2sam phase of mg-aligner ****")

def run_sam_pad(namespace: str, release: str):
    api_responses = create_job_resources(
        namespace,
        release,
        "sam-pad",
        f"bwbble/mg-ref:{bwbble_container_image_version}",
        args=[
            f"/mg-ref-output/{bubble_file}",
            f"/mg-align-output/{release}.aligned_reads.sam",
            f"/mg-align-output/{release}.output.sam",
        ],
        env=[kubernetes.client.V1EnvVar(name="APPLICATION", value="sam_pad"),],
    )

    # Wait for the sam_pad job to complete
    wait_for_all_jobs(
        namespace,
        release,
        "sam-pad",
        [r for r in api_responses if isinstance(r, kubernetes.client.V1Job)],
    )
    print("All Jobs completed for sam_pad phase of mg-ref")

def kube_test_credentials():

    try:
        api_response = api_instance.get_api_resources()
    except ApiException as e:
        print("Exception when calling API: %s\n" % e)
        sys.exit(0)

def main():
    kube_test_credentials()
    print("**** Done testing credentials ****")
    time_stamp = time.strftime("%H-%M", time.localtime())
```

```
    release = f"test-t{time_stamp}-p{parallelism}-flarge"

    # run_index("bwbble-dev", "test-"+time_stamp)
    run_align("bwbble-dev", release)
    execution_times("bwbble-dev", release, "align", align_logs=True)
    execution_times("bwbble-dev", release, "merge")
    execution_times("bwbble-dev", release, "aln2sam")
    execution_times("bwbble-dev", release, "sam-pad")

if __name__ == "__main__":
    main()
```

Appendix 10 controller/k8sbwbble/jobs/execution_time_job.py

```
class ExecutionTimeJob(Job):
    def __init__(self, stage: str):
        super().__init__(stage)

    def run(self, job: V1AlignJob):
        job.status.execution_times[self.stage] = {}

        try:

            # get execution time for pods
            api_response = self.api_instance.list_namespaced_job(
                namespace=job.metadata.namespace,
                label_selector=f"bwbble-release={job.metadata.name},bwbble-stage={self.stage}",
            )

            for item in api_response.items:
                job.status.execution_times[self.stage][item.metadata.name] = {
                    "total": -1
                }

                if item.status.completion_time:
                    execution_time = (
                        item.status.completion_time - item.status.start_time
                    )
                    print(
                        item.metadata.name, " (job): ", execution_time,
                    )

                    job.status.execution_times[self.stage][item.metadata.name][
                        "total"
                    ] = str(execution_time)
                else:
                    print(item.metadata.name, " (job): Not yet finished")

            if self.stage == "align":
                # get execution time for pods
                api_response = CoreV1Api(self.api_client).list_namespaced_pod(
                    namespace=job.metadata.namespace,
                    label_selector=f"bwbble-release={job.metadata.name},bwbble-stage={self.stage}",
                )
                for item in api_response.items:
                    logs = CoreV1Api(self.api_client).read_namespaced_pod_log(
                        item.metadata.name,
                        job.metadata.namespace,
                        container="align",
                        tail_lines=100,
                    )
                    with open(
                        f"{job.metadata.namespace}-{item.metadata.name}.log", "w+"
                    ) as f:
                        if logs:
                            f.write(logs)
                            f.close()
                            rem = re.search(
                                r"read alignment time: (\d+\.?\d*) sec",
                                logs,
                                re.IGNORECASE,
                            )

                            if rem:
                                print(
                                    item.metadata.name,
                                    " (logs): ",
                                    timedelta(seconds=float(rem[1])),
                                )

                                job.status.execution_times[self.stage][
                                    item.metadata.labels["job-name"]
                                ]["internal"] = str(timedelta(seconds=float(rem[1])))
        except ApiException as e:
            print(e)
```

Appendix 11 controller/crd.yaml

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below and be in the form: <plural>.<group>
  name: alignjobs.bwbble.aideen.dev
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: bwbble.aideen.dev
  # list of versions supported by this CustomResourceDefinition
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage version.
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              required: ["readsFile", "readsCount", "bubbleFile", "snpFile"]
              properties:
                readsFile:
                  type: string
                  description: "The name of the file from which DNA reads will be consumed.
"
                bwbbleVersion:
                  type: string
                  description: "The version of the bwbble toolchain to use."
                  default: "latest"
                readsCount:
                  type: integer
                  description: "The number of reads in the reads file."
                alignParallelism:
                  type: integer
                  description: "The number of containers that will each run a section of th
e reads file."
                  default: 1
                bubbleFile:
                  type: string
                  description: "The reference genome bubble data used in aln2sam to create
aligned reads .SAM file."
                snpFile:
                  type: string
                  description: "The multireference genome file for short-
read alignment of the reads file."
            status:
              type: object
              properties:
                startTime:
                  type: string
                endTime:
                  type: string
                stage:
                  type: string
```

```
                waitingFor:
                  type: array
                  items:
                    type: string
                  default: []
                executionTimes:
                  type: object
                  additionalProperties:
                    type: object
                    additionalProperties:
                      type: object
                      required:
                        - total
                      properties:
                        total:
                          type: string
                          description: "The total end-to-
end execution time measured by the Kubernetes API"
                        internal:
                          type: string
                          description: "An internally measured execution time from the appl
ication."

  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: alignjobs
    # singular name to be used as an alias on the CLI and for display
    singular: alignjob
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: AlignJob
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
      - aj
      - ajs
```

Appendix 12 controller/k8sbwbble/job_spec.py

```
from kubernetes import client
import six
import pprint
import re  # noqa: F401
from typing import List
from datetime import datetime

class V1AlignJobSpec(object):
    openapi_types = {
        "reads_file": "str",
        "reads_count": "int",
        "bubble_file": "str",
        "snp_file": "str",
        "align_parallelism": "int",
        "bwbble_version": "str",
    }

    attribute_map = {
        "reads_file": "readsFile",
        "reads_count": "readsCount",
        "bubble_file": "bubbleFile",
        "snp_file": "snpFile",
        "align_parallelism": "alignParallelism",
        "bwbble_version": "bwbbleVersion",
    }

    def __init__(
        self,
        reads_file: str = None,
        reads_count: int = 512000,
        align_parallelism: int = 1,
        bwbble_version: int = "latest",
        bubble_file: str = None,
        snp_file: str = None,
        **kwags
    ):
        self.reads_count = reads_count
        self.reads_file = reads_file
        self.bubble_file = bubble_file
        self.snp_file = snp_file
        self.align_parallelism = align_parallelism
        self.bwbble_version = bwbble_version

    def to_dict(self):
        """Returns the model properties as a dict"""
        result = {}

        for attr, _ in six.iteritems(self.openapi_types):
            value = getattr(self, attr)
            if isinstance(value, list):
                result[attr] = list(
                    map(lambda x: x.to_dict() if hasattr(x, "to_dict") else x, value)
                )
            elif hasattr(value, "to_dict"):
                result[attr] = value.to_dict()
            elif isinstance(value, dict):
```

```
                result[attr] = dict(
                    map(
                        lambda item: (item[0], item[1].to_dict())
                        if hasattr(item[1], "to_dict")
                        else item,
                        value.items(),
                    )
                )
            else:
                result[attr] = value

        return result

    def to_str(self):
        """Returns the string representation of the model"""
        return pprint.pformat(self.to_dict())

    def __repr__(self):
        """For `print` and `pprint`"""
        return self.to_str()

    def __eq__(self, other):
        """Returns true if both objects are equal"""
        if not isinstance(other, V1AlignJobSpec):
            return False

        return self.__dict__ == other.__dict__

    def __ne__(self, other):
        """Returns true if both objects are not equal"""
        return not self == other

class V1AlignJobStatus(object):
    openapi_types = {
        "stage": "str",
        "waiting_for": "list[str]",
        "start_time": "datetime",
        "end_time": "datetime",
        "execution_times": "dict(str, dict(str, dict(str, str)))",
    }

    attribute_map = {
        "stage": "stage",
        "waiting_for": "waitingFor",
        "start_time": "startTime",
        "end_time": "endTime",
        "execution_times": "executionTimes",
    }

    def __init__(
        self,
        stage: str = None,
        waiting_for: List[str] = None,
        start_time: datetime = None,
        end_time: datetime = None,
        execution_times: dict = None,
        **kwags
    ):
```

```
        self.stage = stage
        self.waiting_for = waiting_for or []
        self.start_time = start_time
        self.end_time = end_time
        self.execution_times = execution_times or {}

    def to_dict(self):
        """Returns the model properties as a dict"""
        result = {}

        for attr, _ in six.iteritems(self.openapi_types):
            value = getattr(self, attr)
            if isinstance(value, list):
                result[attr] = list(
                    map(lambda x: x.to_dict() if hasattr(x, "to_dict") else x, value)
                )
            elif hasattr(value, "to_dict"):
                result[attr] = value.to_dict()
            elif isinstance(value, dict):
                result[attr] = dict(
                    map(
                        lambda item: (item[0], item[1].to_dict())
                        if hasattr(item[1], "to_dict")
                        else item,
                        value.items(),
                    )
                )
            else:
                result[attr] = value

        return result

    def to_str(self):
        """Returns the string representation of the model"""
        return pprint.pformat(self.to_dict())

    def __repr__(self):
        """For `print` and `pprint`"""
        return self.to_str()

    def __eq__(self, other):
        """Returns true if both objects are equal"""
        if not isinstance(other, V1AlignJobStatus):
            return False

        return self.__dict__ == other.__dict__

    def __ne__(self, other):
        """Returns true if both objects are not equal"""
        return not self == other

class V1AlignJob(object):
    openapi_types = {
        "api_version": "str",
        "kind": "str",
        "metadata": "V1ObjectMeta",
        "spec": "V1AlignJobSpec",
        "status": "V1AlignJobStatus",
```

```
}

attribute_map = {
    "api_version": "apiVersion",
    "kind": "kind",
    "metadata": "metadata",
    "spec": "spec",
    "status": "status",
}

def __init__(
    self,
    metadata: client.V1ObjectMeta = None,
    spec: V1AlignJobSpec = None,
    status: V1AlignJobStatus = None,
    **kwags
):
    super().__init__()

    self.metadata = metadata or client.V1ObjectMeta()
    self.spec = spec or V1AlignJobSpec()
    self.status = status or V1AlignJobStatus()

@property
def api_version(self):
    return "bwbble.aideen.dev/v1"

@property
def kind(self):
    return "AlignJob"

def to_dict(self):
    """Returns the model properties as a dict"""
    result = {}

    for attr, _ in six.iteritems(self.openapi_types):
        value = getattr(self, attr)
        if isinstance(value, list):
            result[attr] = list(
                map(lambda x: x.to_dict() if hasattr(x, "to_dict") else x, value)
            )
        elif hasattr(value, "to_dict"):
            result[attr] = value.to_dict()
        elif isinstance(value, dict):
            result[attr] = dict(
                map(
                    lambda item: (item[0], item[1].to_dict())
                    if hasattr(item[1], "to_dict")
                    else item,
                    value.items(),
                )
            )
        else:
            result[attr] = value

    return result

def to_str(self):
```

```
        """Returns the string representation of the model"""
        return pprint.pformat(self.to_dict())

    def __repr__(self):
        """For `print` and `pprint`"""
        return self.to_str()

    def __eq__(self, other):
        """Returns true if both objects are equal"""
        if not isinstance(other, V1AlignJob):
            return False

        return self.__dict__ == other.__dict__

    def __ne__(self, other):
        """Returns true if both objects are not equal"""
        return not self == other

def add_to_api_client(api_client: client.ApiClient):
    api_client.NATIVE_TYPES_MAPPING["V1AlignJob"] = V1AlignJob
    api_client.NATIVE_TYPES_MAPPING["V1AlignJobSpec"] = V1AlignJobSpec
    api_client.NATIVE_TYPES_MAPPING["V1AlignJobStatus"] = V1AlignJobStatus
```

Appendix 13 controller/app/main.py

```
from k8sbwbble.controller import Controller
from kubernetes.config import load_kube_config, load_incluster_config

if __name__ == "__main__":
    print("Configuring bwbble controller for access to K8s API")

    # Load the Kubernetes config file
    try:
        load_incluster_config()
    except:
        load_kube_config()

    controller = Controller()

    print("Starting controller for namespace bwbble-dev")
    controller.run("bwbble-dev")
```

Appendix 14 controller/deployment.yaml

```yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bwbble-controller
  namespace: bwbble-dev
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: bwbble-dev
  name: bwbble-controller
rules:
  - apiGroups: ["bwbble.aideen.dev"]
    resources: ["alignjobs"]
    verbs: ["*"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "watch", "list", "create", "update", "delete"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "create", "update", "delete"]
  - apiGroups: [""]
    resources: ["pods", "pods/log"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bwbble-controller
  namespace: bwbble-dev
subjects:
  - kind: ServiceAccount
    name: bwbble-controller
    namespace: bwbble-dev
roleRef:
  kind: Role
  name: bwbble-controller
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bwbble-controller
  namespace: bwbble-dev
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bwbble-controller
  template:
    metadata:
      labels:
        app: bwbble-controller
    spec:
      containers:
```

```
    - name: controller
      image: bwbble/controller:16
      args:
        - "--namespace=bwbble-dev"

  serviceAccountName: bwbble-controller
```

Appendix 15 controller/app/benchmark.py

```
#!/usr/bin/env python
import csv
import statistics
from k8sbwbble.controller import Controller
from k8sbwbble.job_spec import V1AlignJob, V1AlignJobSpec
from kubernetes.client import V1ObjectMeta, CustomObjectsApi
from kubernetes.config import load_kube_config, load_incluster_config
from kubernetes.watch import Watch
from datetime import timedelta
import re

parse_timedelta_regex = re.compile(
    r"(?P<hours>\d+):(?P<minutes>\d+):(?P<seconds>\d+)(.(?P<microseconds>\d+))?"
)

def parse_timedelta(input: str) -> timedelta:
    parts = parse_timedelta_regex.match(input)
    if not parts:
        return
    parts = parts.groupdict()
    time_params = {}
    for (name, param) in parts.items():
        if param:
            time_params[name] = int(param)
    return timedelta(**time_params)

def main():
    print("Configuring bwbble controller for access to K8s API")

    benchmark_type = "large_reads"

    # Load the Kubernetes config file
    try:
        load_incluster_config()
    except:
        load_kube_config()

    controller = Controller()
    with open("benchmarking.csv", mode="w", buffering=1) as bench_file:
        bench_writer = csv.writer(
            bench_file, delimiter=",", quotechar='"', quoting=csv.QUOTE_MINIMAL
        )

        fields = [
            "total_runtime",
            "align_time",
            "align_job",
            "merge_job",
            "aln2sam_job",
            "sampad_job",
        ]

        bench_writer.writerow(
            [
                "no_containers",
                *[f"{field}_mean" for field in fields],
```

```
                *[f"{field}_median" for field in fields],
                *[f"{field}_min" for field in fields],
                *[f"{field}_max" for field in fields],
            ]
        )

        run_stats = {}
        for parallelism in range(1, 17):

            align_times = []
            align_job_times = []
            merge_job_times = []
            aln2sam_job_times = []
            sampad_job_times = []
            runtimes = []

            for run_index in range(1, 5):
                print(f"Running index {run_index} with parallelism {parallelism}")
                job = V1AlignJob(
                    metadata=V1ObjectMeta(
                        name=f"bench-{benchmark_type}-p{parallelism}-r{run_index}",
                        namespace="bwbble-dev",
                    ),
                    spec=V1AlignJobSpec(
                        align_parallelism=parallelism,
                        reads_count=512000,
                        reads_file="dummy_reads.fastq",
                        bubble_file="chr21_bubble.data",
                        snp_file="chr21_ref_w_snp_and_bubble.fasta",
                        bwbble_version="313",
                    ),
                )

                try:
                    controller.create_align_job(job)
                except Exception as ex:
                    print(
                        f"Failed to create V1AlignJob for parallelism={parallelism}, iterat
ion={run_index}"
                    )

                watcher = Watch(return_type=V1AlignJob)
                for event in watcher.stream(
                    CustomObjectsApi(
                        controller.api_client
                    ).list_namespaced_custom_object,
                    "bwbble.aideen.dev",
                    "v1",
                    job.metadata.namespace,
                    "alignjobs",
                ):
                    if event["object"].metadata.name == job.metadata.name:
                        if event["object"].status.end_time is not None:
                            watcher.stop()
                            run_stats[job.metadata.name] = event["object"].status

                            runtimes.append(
                                (
```

```
                                event["object"].status.end_time
                                - event["object"].status.start_time
                            ).total_seconds()
                        )

                        for tag, stats in {
                            "align": align_job_times,
                            "merge": merge_job_times,
                            "aln2sam": aln2sam_job_times,
                            "sampad": sampad_job_times,
                        }.items():
                            for job_id, times in (
                                event["object"].status.execution_times[tag].items()
                            ):
                                stats.append(
                                    parse_timedelta(times["total"]).total_seconds()
                                )

                        for job_id, times in (
                            event["object"].status.execution_times["align"].items()
                        ):
                            align_times.append(
                                parse_timedelta(times["internal"]).total_seconds()
                            )

            stats = [
                runtimes,
                align_times,
                align_job_times,
                merge_job_times,
                aln2sam_job_times,
                sampad_job_times,
            ]

            bench_writer.writerow(
                [
                    parallelism,
                    *[timedelta(seconds=statistics.mean(s)) for s in stats],
                    *[timedelta(seconds=statistics.median(s)) for s in stats],
                    *[timedelta(seconds=min(s)) for s in stats],
                    *[timedelta(seconds=max(s)) for s in stats],
                ]
            )

if __name__ == "__main__":
    main()
```

Appendix 16 Controller Source Code

```
####################
# app/benchmark.py #
####################

#!/usr/bin/env python
import csv
import statistics
```

```
from k8sbwbble.controller import Controller
from k8sbwbble.job_spec import V1AlignJob, V1AlignJobSpec
from kubernetes.client import V1ObjectMeta, CustomObjectsApi
from kubernetes.config import load_kube_config, load_incluster_config
from kubernetes.watch import Watch
from datetime import timedelta
import re

parse_timedelta_regex = re.compile(
    r"(?P<hours>\d+):(?P<minutes>\d+):(?P<seconds>\d+)(.(?P<microseconds>\d+))?"
)

def parse_timedelta(input: str) -> timedelta:
    parts = parse_timedelta_regex.match(input)
    if not parts:
        return
    parts = parts.groupdict()
    time_params = {}
    for (name, param) in parts.items():
        if param:
            time_params[name] = int(param)
    return timedelta(**time_params)

def main():
    print("Configuring bwbble controller for access to K8s API")

    benchmark_type = "noaci1"

    # Load the Kubernetes config file
    try:
        load_incluster_config()
    except:
        load_kube_config()

    controller = Controller()
    with open("benchmarking_no_aci.csv", mode="w", buffering=1) as bench_file:
        bench_writer = csv.writer(
            bench_file, delimiter=",", quotechar='"', quoting=csv.QUOTE_MINIMAL
        )

        fields = [
            "total_runtime",
            "align_time",
            "align_job",
            "merge_job",
            "aln2sam_job",
            "sampad_job",
        ]

        bench_writer.writerow(
            [
                "no_containers",
                *[f"{field}_mean" for field in fields],
                *[f"{field}_median" for field in fields],
                *[f"{field}_min" for field in fields],
                *[f"{field}_max" for field in fields],
            ]
        )
```

```
        run_stats = {}
        for parallelism in range(1, 17):

            align_times = []
            align_job_times = []
            merge_job_times = []
            aln2sam_job_times = []
            sampad_job_times = []
            runtimes = []

            for run_index in range(1, 4):
                print(
                    f"Running index {run_index} with parallelism {parallelism}")
                job = V1AlignJob(
                    # TODO: Use argparse to provide the namespace as an argument
                    metadata=V1ObjectMeta(
                        name=f"bench-{benchmark_type}-p{parallelism}-r{run_index}",
                        namespace="bwbble-dev",
                    ),
                    spec=V1AlignJobSpec(
                        align_parallelism=parallelism,
                        reads_count=512000,
                        reads_file="dummy_reads.fastq",
                        bubble_file="chr21_bubble.data",
                        snp_file="chr21_ref_w_snp_and_bubble.fasta",
                        bwbble_version="313",
                    ),
                )

                try:
                    controller.create_align_job(job)
                except Exception as ex:
                    print(
                        f"Failed to create V1AlignJob for parallelism={parallelism}, iterat
ion={run_index}"
                    )

                watcher = Watch(return_type=V1AlignJob)
                for event in watcher.stream(
                    CustomObjectsApi(
                        controller.api_client
                    ).list_namespaced_custom_object,
                    "bwbble.aideen.dev",
                    "v1",
                    job.metadata.namespace,
                    "alignjobs",
                ):
                    if event["object"].metadata.name == job.metadata.name:
                        if event["object"].status.end_time is not None:
                            watcher.stop()
                            run_stats[job.metadata.name] = event["object"].status

                            runtimes.append(
                                (
                                    event["object"].status.end_time
                                    - event["object"].status.start_time
                                ).total_seconds()
```

```
                            )

                            for tag, stats in {
                                "align": align_job_times,
                                "merge": merge_job_times,
                                "aln2sam": aln2sam_job_times,
                                "sampad": sampad_job_times,
                            }.items():
                                for job_id, times in (
                                    event["object"].status.execution_times[tag].items()
                                ):
                                    stats.append(
                                        parse_timedelta(
                                            times["total"]).total_seconds()
                                    )

                            for job_id, times in (
                                event["object"].status.execution_times["align"].items()
                            ):
                                align_times.append(
                                    parse_timedelta(
                                        times["internal"]).total_seconds()
                                )

                    stats = [
                        runtimes,
                        align_times,
                        align_job_times,
                        merge_job_times,
                        aln2sam_job_times,
                        sampad_job_times,
                    ]

                    bench_writer.writerow(
                        [
                            parallelism,
                            *[timedelta(seconds=statistics.mean(s)) for s in stats],
                            *[timedelta(seconds=statistics.median(s)) for s in stats],
                            *[timedelta(seconds=min(s)) for s in stats],
                            *[timedelta(seconds=max(s)) for s in stats],
                        ]
                    )

if __name__ == "__main__":
    main()

###############
# app/main.py #
###############

if __name__ == "__main__":
    print("Configuring bwbble controller for access to K8s API")

    # Load the Kubernetes config file
    try:
        load_incluster_config()
    except:
        load_kube_config()
```

```
    controller = Controller()

    # TODO: Use argparse to provide the namespace as an argument

    print("Starting controller for namespace bwbble-dev")
    controller.run("bwbble-dev")

############################
# k8sbwbble/controller.py #
############################

from .job_spec import V1AlignJob, add_to_api_client
from .jobs.align_job import AlignJob
from .jobs.merge_job import MergeJob
from .jobs.aln2sam_job import Aln2SamJob
from .jobs.sampad_job import SampadJob
from .jobs.execution_time_job import ExecutionTimeJob
from .jobs.cleanup_job import CleanupJob
from threading import Thread
from json import dumps
import datetime

from kubernetes.watch import Watch
from kubernetes.client import (
    ApiClient,
    BatchV1Api,
    Configuration,
    V1Job,
    CustomObjectsApi,
)
from kubernetes.client.rest import RESTResponse
from typing import List

# triggered by save_align_job_state as well as kubectl
class AlignJobWatcherThread(Thread):
    def __init__(self, controller: "Controller", namespace: str):
        super().__init__()

        self.controller = controller
        self.namespace = namespace

    def run(self):
        print("Starting AlignJob watcher thread")
        watcher = Watch(return_type=V1AlignJob)
        for event in watcher.stream(
            CustomObjectsApi(self.controller.api_client).list_namespaced_custom_object,
            "bwbble.aideen.dev",
            "v1",
            self.namespace,
            "alignjobs",
        ):
            self.controller.on_align_job_updated(event["object"], event["type"])

# monitors resources (jobs) created by create_job_resources
class BatchJobWatcherThread(Thread):
    def __init__(self, controller: "Controller", namespace: str):
        super().__init__()
```

```
        self.controller = controller
        self.namespace = namespace

    def run(self):
        print("Starting BatchJob watcher thread")
        watcher = Watch(return_type=V1Job)
        for event in watcher.stream(
            BatchV1Api(self.controller.api_client).list_namespaced_job, self.namespace,
        ):
            # TODO: Ignore deletion and creation events (only trigger on updates)
            self.controller.on_job_updated(event["object"], event["type"])

class MockResponse(object):
    def __init__(self, response):
        super().__init__()
        self.data = dumps(response)

class Controller(object):
    def __init__(self):
        super().__init__()

        self.config = Configuration()
        self.api_client = ApiClient(self.config)
        add_to_api_client(self.api_client)

    def run(self, namespace: str):
        align_job_watcher = AlignJobWatcherThread(self, namespace)
        batch_job_watcher = BatchJobWatcherThread(self, namespace)

        align_job_watcher.start()
        batch_job_watcher.start()

        align_job_watcher.join()
        batch_job_watcher.join()

    def create_align_job(self, job: V1AlignJob):
        return CustomObjectsApi(self.api_client).create_namespaced_custom_object(
            "bwbble.aideen.dev", "v1", job.metadata.namespace, "alignjobs", job,
        )

    def get_align_job(self, namespace: str, name: str) -> V1AlignJob:
        response = CustomObjectsApi(self.api_client).get_namespaced_custom_object(
            "bwbble.aideen.dev", "v1", namespace, "alignjobs", name,
        )

        return self.api_client.deserialize(MockResponse(response), V1AlignJob)

    def save_align_job_state(self, job: V1AlignJob):
        return CustomObjectsApi(self.api_client).replace_namespaced_custom_object(
            "bwbble.aideen.dev",
            "v1",
            job.metadata.namespace,
            "alignjobs",
            job.metadata.name,
            job,
        )
```

```
    def on_job_updated(self, job: V1Job, change_type: str):
        """
        Handles the change of state in a Kubernetes Batch V1 job (the unit of execution
        we work with in K8s).
        """
        print(f"Job {job.metadata.name} {change_type}")

        try:
            if job.status.completion_time is None:
                return

            if "bwbble-alignjob-name" not in job.metadata.labels:
                return

            align_job_id = job.metadata.labels["bwbble-alignjob-name"]
            align_job = self.get_align_job(job.metadata.namespace, align_job_id)
            if not align_job:
                return

            if job.metadata.name in align_job.status.waiting_for:
                align_job.status.waiting_for.remove(job.metadata.name)

                self.save_align_job_state(align_job)
        except Exception as ex:
            # TODO: If this fails for an unrecoverable reason, update the
            #       alignjob to reflect that.
            #       e.g. if a job fails permanently/is deleted
            print(ex)

    def on_align_job_updated(self, job: V1AlignJob, change_type: str):
        """
        Handles the change of state in one of our high-level AlignJobs (a custom resource).
        """
        print(f"AlignJob {job.metadata.name} {change_type}")

        try:
            if len(job.status.waiting_for) > 0:
                print(
                    f"Waiting for {len(job.status.waiting_for)} jobs to complete for {job.s
tatus.stage}"
                )
                return

            if job.status.stage is not None:
                print(f"Finished all jobs for {job.status.stage}")
                ExecutionTimeJob(job.status.stage).run(job)

            if job.status.stage is None:
                # Execute the align job
                job.status.start_time = datetime.datetime.utcnow()
                job.status.stage = "align"
                AlignJob().run(job)
            elif job.status.stage == "align":
                job.status.stage = "merge"
                MergeJob().run(job)
            elif job.status.stage == "merge":
                job.status.stage = "aln2sam"
                Aln2SamJob().run(job)
```

```
            elif job.status.stage == "aln2sam":
                job.status.stage = "sampad"
                SampadJob().run(job)
            elif job.status.stage == "sampad":
                job.status.stage = "cleanup"
                CleanupJob().run(job)
            elif job.status.stage == "cleanup":
                job.status.stage = "completed"
                job.status.end_time = datetime.datetime.utcnow()
            elif job.status.stage == "completed":
                print(
                    f"Finished full alignment of {job.spec.reads_file} with parallelism of
{job.spec.align_parallelism}"
                )
                return

            print(f"Started stage {job.status.stage}")

            self.save_align_job_state(job)
        except Exception as ex:
            print(ex)

            # TODO: If we can't schedule the next phase of work, mark the job as failed wit
h a reason
            #       e.g. there is another job with the same name and conflicting resources
(we get a 409)

############################
# k8sbwbble/file_range.py #
############################

from math import floor

class Range(object):
    def __init__(self, start: int, length: int = -1):
        self.start = start
        self.length = length

    @property
    def name(self) -> str:
        if self.length == -1:
            return f"{self.start}-end"
        return f"{self.start}-{self.start+self.length}"

    @staticmethod
    def generate(num_reads: int, parallelism: int = 1):
        rrange = floor(num_reads / parallelism)
        ranges = []
        for i in range(0, parallelism):
            ranges.append(Range(i * rrange, rrange))

        ranges[len(ranges) - 1].length = -1

        return ranges

####################
# k8sbwbble/job.py #
####################
```

```python
from .job_spec import V1AlignJob
from kubernetes import client, config, utils
from kubernetes.client import (
    ApiClient,
    BatchV1Api,
    Configuration,
    CoreV1Api,
    V1AzureFileVolumeSource,
    V1ConfigMap,
    V1ConfigMapVolumeSource,
    V1Container,
    V1EnvVar,
    V1Job,
    V1JobSpec,
    V1ObjectMeta,
    V1PodSpec,
    V1PodTemplate,
    V1PodTemplateSpec,
    V1ResourceRequirements,
    V1Toleration,
    V1Volume,
    V1VolumeMount,
)
from kubernetes.client.rest import ApiException
from kubernetes.watch import Watch
from typing import List

class Job(object):
    def __init__(self, stage: str):
        super().__init__()
        self.stage = stage

        configuration = Configuration()
        self.api_client = ApiClient(configuration)
        self.api_instance = BatchV1Api(self.api_client)

    def get_job_name(self, job: V1AlignJob, name_suffix: str = "") -> str:
        return f"bwbble-{job.metadata.name}-{self.stage}{name_suffix}"

    def create_job_resources(
        self,
        job: V1AlignJob,
        container_image: str,
        args: List[str],
        use_config_map_args: bool = True,
        resources: V1ResourceRequirements = None,
        env: List[V1EnvVar] = None,
        name_suffix: str = "",
        use_aci: bool = True,
    ):
        labels = {
            # TODO: Remove this once there are no longer dependencies on it
            "bwbble-release": job.metadata.name,
            "bwbble-alignjob-name": job.metadata.name,
            "bwbble-stage": self.stage,
        }
```

```
resources = resources or V1ResourceRequirements(
    limits={"memory": "2Gi", "cpu": "1"}, requests={"memory": "2Gi", "cpu": "1"}
)
created_resources = []
env_array = []

if use_aci and not use_config_map_args:
    raise Exception(
        "Azure Container Instances require that arguments are passed using a file"
    )

if use_config_map_args:
    env_array.append(
        V1EnvVar(name="ARGS_FILE", value="/var/run/args/container_args")
    )

    config_map = V1ConfigMap(
        api_version="v1",
        kind="ConfigMap",
        metadata=V1ObjectMeta(
            name=self.get_job_name(job, name_suffix), labels=labels,
        ),
        data={
            "container_args": " ".join(
                [f"'{a}'" if " " in a else a for a in args]
            ),
        },
    )

    try:
        created_resources.append(
            CoreV1Api(self.api_client).create_namespaced_config_map(
                job.metadata.namespace, config_map
            )
        )
    except ApiException as ex:
        if ex.status == 409:
            created_resources.append(
                CoreV1Api(self.api_client).replace_namespaced_config_map(
                    self.get_job_name(job, name_suffix),
                    job.metadata.namespace,
                    config_map,
                )
            )
        else:
            raise ex

if env is not None:
    env_array.extend(env)

job_spec = V1Job(
    api_version="batch/v1",
    kind="Job",
    metadata=V1ObjectMeta(
        name=self.get_job_name(job, name_suffix), labels=labels,
    ),
    spec=V1JobSpec(
        template=V1PodTemplateSpec(
```

```
            metadata=V1ObjectMeta(name=self.stage, labels=labels),
            spec=V1PodSpec(
                restart_policy="Never",
                containers=[
                    V1Container(
                        name=self.stage,
                        image=container_image,
                        image_pull_policy="IfNotPresent",
                        args=args,
                        env=env_array,
                        resources=resources,
                        volume_mounts=[
                            V1VolumeMount(mount_path="/input", name="input"),
                            V1VolumeMount(
                                mount_path="/mg-ref-output", name="ref-output"
                            ),
                            V1VolumeMount(
                                mount_path="/mg-align-output",
                                name="align-output",
                            ),
                        ],
                    )
                ],
                volumes=[
                    V1Volume(
                        name="input",
                        azure_file=V1AzureFileVolumeSource(
                            secret_name="azure-secret",
                            share_name="input",
                            read_only=True,
                        ),
                    ),
                    V1Volume(
                        name="ref-output",
                        azure_file=V1AzureFileVolumeSource(
                            secret_name="azure-secret",
                            share_name="ref-output",
                            read_only=False,
                        ),
                    ),
                    V1Volume(
                        name="align-output",
                        azure_file=V1AzureFileVolumeSource(
                            secret_name="azure-secret",
                            share_name="align-output",
                            read_only=False,
                        ),
                    ),
                ],
            ),
        )
    ),
)

if use_aci:
    job_spec.spec.template.spec.node_selector = {
        "kubernetes.io/role": "agent",
        "beta.kubernetes.io/os": "linux",
```

```
                "type": "virtual-kubelet",
            }

            job_spec.spec.template.spec.tolerations = [
                V1Toleration(key="virtual-kubelet.io/provider", operator="Exists"),
                V1Toleration(key="azure.com/aci", effect="NoSchedule"),
            ]

        if use_config_map_args:

            job_spec.spec.template.spec.volumes.append(
                V1Volume(
                    name="args",
                    config_map=V1ConfigMapVolumeSource(
                        name=f"bwbble-{job.metadata.name}-{self.stage}{name_suffix}"
                    ),
                )
            )

            job_spec.spec.template.spec.containers[0].args = []
            job_spec.spec.template.spec.containers[0].volume_mounts.append(
                V1VolumeMount(mount_path="/var/run/args", name="args")
            )

        try:
            created_resources.append(
                BatchV1Api(self.api_client).create_namespaced_job(
                    job.metadata.namespace, job_spec
                )
            )
        except ApiException as ex:
            if ex.status == 409:
                created_resources.append(
                    BatchV1Api(self.api_client).replace_namespaced_job(
                        job.metadata.name, job.metadata.namespace, job_spec
                    )
                )
            else:
                raise ex

        return created_resources

    def run(self, job: V1AlignJob):
        raise Exception("Not yet implemented")

#########################
# k8sbwbble/job_spec.py #
#########################

from kubernetes import client
import six
import pprint
import re  # noqa: F401
from typing import List
from datetime import datetime

class V1AlignJobSpec(object):
    openapi_types = {
```

```
    "reads_file": "str",
    "reads_count": "int",
    "bubble_file": "str",
    "snp_file": "str",
    "align_parallelism": "int",
    "bwbble_version": "str",
}

attribute_map = {
    "reads_file": "readsFile",
    "reads_count": "readsCount",
    "bubble_file": "bubbleFile",
    "snp_file": "snpFile",
    "align_parallelism": "alignParallelism",
    "bwbble_version": "bwbbleVersion",
}

def __init__(
    self,
    reads_file: str = None,
    reads_count: int = 512000,
    align_parallelism: int = 1,
    bwbble_version: int = "latest",
    bubble_file: str = None,
    snp_file: str = None,
    **kwags
):
    self.reads_count = reads_count
    self.reads_file = reads_file
    self.bubble_file = bubble_file
    self.snp_file = snp_file
    self.align_parallelism = align_parallelism
    self.bwbble_version = bwbble_version

def to_dict(self):
    """Returns the model properties as a dict"""
    result = {}

    for attr, _ in six.iteritems(self.openapi_types):
        value = getattr(self, attr)
        if isinstance(value, list):
            result[attr] = list(
                map(lambda x: x.to_dict() if hasattr(x, "to_dict") else x, value)
            )
        elif hasattr(value, "to_dict"):
            result[attr] = value.to_dict()
        elif isinstance(value, dict):
            result[attr] = dict(
                map(
                    lambda item: (item[0], item[1].to_dict())
                    if hasattr(item[1], "to_dict")
                    else item,
                    value.items(),
                )
            )
        else:
            result[attr] = value
```

```
        return result

    def to_str(self):
        """Returns the string representation of the model"""
        return pprint.pformat(self.to_dict())

    def __repr__(self):
        """For `print` and `pprint`"""
        return self.to_str()

    def __eq__(self, other):
        """Returns true if both objects are equal"""
        if not isinstance(other, V1AlignJobSpec):
            return False

        return self.__dict__ == other.__dict__

    def __ne__(self, other):
        """Returns true if both objects are not equal"""
        return not self == other

class V1AlignJobStatus(object):
    openapi_types = {
        "stage": "str",
        "waiting_for": "list[str]",
        "start_time": "datetime",
        "end_time": "datetime",
        "execution_times": "dict(str, dict(str, dict(str, str)))",
    }

    attribute_map = {
        "stage": "stage",
        "waiting_for": "waitingFor",
        "start_time": "startTime",
        "end_time": "endTime",
        "execution_times": "executionTimes",
    }

    def __init__(
        self,
        stage: str = None,
        waiting_for: List[str] = None,
        start_time: datetime = None,
        end_time: datetime = None,
        execution_times: dict = None,
        **kwags
    ):
        self.stage = stage
        self.waiting_for = waiting_for or []
        self.start_time = start_time
        self.end_time = end_time
        self.execution_times = execution_times or {}

    def to_dict(self):
        """Returns the model properties as a dict"""
        result = {}

        for attr, _ in six.iteritems(self.openapi_types):
```

```
            value = getattr(self, attr)
            if isinstance(value, list):
                result[attr] = list(
                    map(lambda x: x.to_dict() if hasattr(x, "to_dict") else x, value)
                )
            elif hasattr(value, "to_dict"):
                result[attr] = value.to_dict()
            elif isinstance(value, dict):
                result[attr] = dict(
                    map(
                        lambda item: (item[0], item[1].to_dict())
                        if hasattr(item[1], "to_dict")
                        else item,
                        value.items(),
                    )
                )
            else:
                result[attr] = value

        return result

    def to_str(self):
        """Returns the string representation of the model"""
        return pprint.pformat(self.to_dict())

    def __repr__(self):
        """For `print` and `pprint`"""
        return self.to_str()

    def __eq__(self, other):
        """Returns true if both objects are equal"""
        if not isinstance(other, V1AlignJobStatus):
            return False

        return self.__dict__ == other.__dict__

    def __ne__(self, other):
        """Returns true if both objects are not equal"""
        return not self == other

class V1AlignJob(object):
    openapi_types = {
        "api_version": "str",
        "kind": "str",
        "metadata": "V1ObjectMeta",
        "spec": "V1AlignJobSpec",
        "status": "V1AlignJobStatus",
    }

    attribute_map = {
        "api_version": "apiVersion",
        "kind": "kind",
        "metadata": "metadata",
        "spec": "spec",
        "status": "status",
    }

    def __init__(
```

```
    self,
    metadata: client.V1ObjectMeta = None,
    spec: V1AlignJobSpec = None,
    status: V1AlignJobStatus = None,
    **kwags
):
    super().__init__()

    self.metadata = metadata or client.V1ObjectMeta()
    self.spec = spec or V1AlignJobSpec()
    self.status = status or V1AlignJobStatus()

@property
def api_version(self):
    return "bwbble.aideen.dev/v1"

@property
def kind(self):
    return "AlignJob"

def to_dict(self):
    """Returns the model properties as a dict"""
    result = {}

    for attr, _ in six.iteritems(self.openapi_types):
        value = getattr(self, attr)
        if isinstance(value, list):
            result[attr] = list(
                map(lambda x: x.to_dict() if hasattr(x, "to_dict") else x, value)
            )
        elif hasattr(value, "to_dict"):
            result[attr] = value.to_dict()
        elif isinstance(value, dict):
            result[attr] = dict(
                map(
                    lambda item: (item[0], item[1].to_dict())
                    if hasattr(item[1], "to_dict")
                    else item,
                    value.items(),
                )
            )
        else:
            result[attr] = value

    return result

def to_str(self):
    """Returns the string representation of the model"""
    return pprint.pformat(self.to_dict())

def __repr__(self):
    """For `print` and `pprint`"""
    return self.to_str()

def __eq__(self, other):
    """Returns true if both objects are equal"""
    if not isinstance(other, V1AlignJob):
        return False
```

```
        return self.__dict__ == other.__dict__

    def __ne__(self, other):
        """Returns true if both objects are not equal"""
        return not self == other

def add_to_api_client(api_client: client.ApiClient):
    api_client.NATIVE_TYPES_MAPPING["V1AlignJob"] = V1AlignJob
    api_client.NATIVE_TYPES_MAPPING["V1AlignJobSpec"] = V1AlignJobSpec
    api_client.NATIVE_TYPES_MAPPING["V1AlignJobStatus"] = V1AlignJobStatus

#########################
# k8sbwbble/__init__.py #
#########################

from .job_spec import V1AlignJob, V1AlignJobSpec, V1AlignJobStatus
from .controller import Controller

##############################
# k8sbwbble/jobs/align_job.py #
##############################

from ..job import V1AlignJob, Job
from ..file_range import Range
from kubernetes.client import V1Job, V1ResourceRequirements

class AlignJob(Job):
    def __init__(self):
        super().__init__("align")

    def run(self, job: V1AlignJob):
        file_ranges = Range.generate(job.spec.reads_count, job.spec.align_parallelism)

        for range in file_ranges:
            api_responses = self.create_job_resources(
                job,
                f"bwbble/mg-aligner:{job.spec.bwbble_version}",
                args=[
                    "align",
                    "-s",
                    f"{range.start}",
                    "-p",
                    f"{range.length}",
                    f"/mg-ref-output/{job.spec.snp_file}",
                    f"/input/{job.spec.reads_file}",
                    f"/mg-align-output/{job.metadata.name}.aligned_reads.{range.name}.aln",
                ],
                name_suffix=f"-{range.name}",
                resources=V1ResourceRequirements(
                    limits={"cpu": "1", "memory": "2Gi"},
                    requests={"cpu": "1", "memory": "2Gi"},
                ),
            )

            job.status.waiting_for.extend(
                [r.metadata.name for r in api_responses if isinstance(r, V1Job)]
            )
```

```
################################
# k8sbwbble/jobs/aln2sam_job.py #
################################

from ..job import V1AlignJob, Job
from kubernetes.client import V1Job

class Aln2SamJob(Job):
    def __init__(self):
        super().__init__("aln2sam")

    def run(self, job: V1AlignJob):
        api_responses = self.create_job_resources(
            job,
            f"bwbble/mg-aligner:{job.spec.bwbble_version}",
            args=[
                "aln2sam",
                f"/mg-ref-output/{job.spec.snp_file}",
                f"/input/{job.spec.reads_file}",
                f"/mg-align-output/{job.metadata.name}.aligned_reads.aln",
                f"/mg-align-output/{job.metadata.name}.aligned_reads.sam",
            ],
        )

        job.status.waiting_for.extend(
            [r.metadata.name for r in api_responses if isinstance(r, V1Job)]
        )

################################
# k8sbwbble/jobs/cleanup_job.py #
################################

from ..job import V1AlignJob, Job
from kubernetes.client import V1Job, CoreV1Api
import time

class CleanupJob(Job):
    def __init__(self):
        super().__init__("cleanup")

    def run(self, job: V1AlignJob):
        self.api_instance.delete_collection_namespaced_job(
            job.metadata.namespace,
            label_selector=f"bwbble-alignjob-name={job.metadata.name}",
            propagation_policy="Foreground",
        )

        CoreV1Api(self.api_client).delete_collection_namespaced_config_map(
            job.metadata.namespace,
            label_selector=f"bwbble-alignjob-name={job.metadata.name}",
        )

##############################
# k8sbwbble/jobs/comb_job.py #
##############################

from ..job import V1AlignJob, Job
```

```
from kubernetes.client import V1Job

class CombJob(Job):
    def __init__(self):
        super().__init__("comb")

    def container_image(self):
        return "bwbble/mg-ref"

    def run(self, job: V1AlignJob):
        # Do the combine job
        api_responses = self.create_job_resources(
            job, f"bwbble/mg-ref:{job.spec.bwbble_version}"
        )

        job.status.waiting_for.extend(
            [r.metadata.name for r in api_responses if isinstance(r, V1Job)]
        )

##################################
# k8sbwbble/jobs/dataprep_job.py #
##################################

from ..job import V1AlignJob, Job
from kubernetes.client import V1Job

class DataPrepJob(Job):
    def __init__(self):
        super().__init__("data-prep")

    def container_image(self):
        return "bwbble/mg-ref"

    def run(self, job: V1AlignJob):
        # Do the dataprep job
        api_responses = self.create_job_resources(
            job, f"bwbble/mg-ref:{job.spec.bwbble_version}"
        )

        job.status.waiting_for.extend(
            [r.metadata.name for r in api_responses if isinstance(r, V1Job)]
        )

########################################
# k8sbwbble/jobs/execution_time_job.py #
########################################

from ..job import V1AlignJob, Job
from _datetime import timedelta
from kubernetes.client import CoreV1Api
from kubernetes.client.rest import ApiException
import re

class ExecutionTimeJob(Job):
    def __init__(self, stage: str):
        super().__init__(stage)

    def run(self, job: V1AlignJob):
```

```
        job.status.execution_times[self.stage] = {}

        try:

            # get execution time for pods
            api_response = self.api_instance.list_namespaced_job(
                namespace=job.metadata.namespace,
                label_selector=f"bwbble-release={job.metadata.name},bwbble-
stage={self.stage}",
            )

            for item in api_response.items:
                job.status.execution_times[self.stage][item.metadata.name] = {
                    "total": -1
                }

                if item.status.completion_time:
                    execution_time = (
                        item.status.completion_time - item.status.start_time
                    )
                    print(
                        item.metadata.name, " (job): ", execution_time,
                    )

                    job.status.execution_times[self.stage][item.metadata.name][
                        "total"
                    ] = str(execution_time)
                else:
                    print(item.metadata.name, " (job): Not yet finished")

            if self.stage == "align":
                # get execution time for pods
                api_response = CoreV1Api(self.api_client).list_namespaced_pod(
                    namespace=job.metadata.namespace,
                    label_selector=f"bwbble-release={job.metadata.name},bwbble-
stage={self.stage}",
                )
                for item in api_response.items:
                    logs = CoreV1Api(self.api_client).read_namespaced_pod_log(
                        item.metadata.name,
                        job.metadata.namespace,
                        container="align",
                        tail_lines=100,
                    )
                    with open(
                        f"{job.metadata.namespace}-{item.metadata.name}.log", "w+"
                    ) as f:
                        if logs:
                            f.write(logs)
                            f.close()
                            rem = re.search(
                                r"read alignment time: (\d+\.?\d*) sec",
                                logs,
                                re.IGNORECASE,
                            )

                            if rem:
                                print(
```

```
                                    item.metadata.name,
                                    " (logs): ",
                                    timedelta(seconds=float(rem[1])),
                                )

                                job.status.execution_times[self.stage][
                                    item.metadata.labels["job-name"]
                                ]["internal"] = str(timedelta(seconds=float(rem[1])))
        except ApiException as e:
            print(e)

##############################
# k8sbwbble/jobs/index_job.py #
##############################

from ..job import V1AlignJob, Job
from kubernetes.client import V1ResourceRequirements, V1Job

class IndexJob(Job):
    def __init__(self):
        super().__init__("index")

    def run(self, job: V1AlignJob):
        api_responses = self.create_job_resources(
            job,
            f"bwbble/mg-aligner:{job.spec.bwbble_version}",
            args=["index", f"/mg-ref-output/{job.spec.snp_file}"],
            resources=V1ResourceRequirements(
                limits={"memory": "2Gi", "cpu": "1"},
                requests={"memory": "2Gi", "cpu": "1"},
            ),
        )

        job.status.waiting_for.extend(
            [r.metadata.name for r in api_responses if isinstance(r, V1Job)]
        )

##############################
# k8sbwbble/jobs/merge_job.py #
##############################

from ..job import V1AlignJob, Job
from ..file_range import Range
from kubernetes.client import V1ResourceRequirements, V1Job

class MergeJob(Job):
    def __init__(self):
        super().__init__("merge")

    def run(self, job: V1AlignJob):

        file_ranges = Range.generate(job.spec.reads_count, job.spec.align_parallelism)
        merge_command = [
            "cat",
            *[
                f"/mg-align-output/{job.metadata.name}.aligned_reads.{range.name}.aln"
                for range in file_ranges
            ],
```

```
            ">",
            f"/mg-align-output/{job.metadata.name}.aligned_reads.aln",
        ]

        api_responses = self.create_job_resources(
            job,
            "busybox:latest",
            use_config_map_args=False,
            use_aci=False,
            args=[
                "sh",
                "-c",
                " ".join([f"'{p}'" if " " in p else p for p in merge_command]),
            ],
            resources=V1ResourceRequirements(
                limits={"memory": "128Mi", "cpu": "1"},
                requests={"memory": "128Mi", "cpu": "50m"},
            ),
        )

        job.status.waiting_for.extend(
            [r.metadata.name for r in api_responses if isinstance(r, V1Job)]
        )

################################
# k8sbwbble/jobs/sampad_job.py #
################################

from ..job import V1AlignJob, Job
from kubernetes.client import V1EnvVar, V1Job

class SampadJob(Job):
    def __init__(self):
        super().__init__("sampad")

    def run(self, job: V1AlignJob):
        api_responses = self.create_job_resources(
            job,
            f"bwbble/mg-ref:{job.spec.bwbble_version}",
            args=[
                f"/mg-ref-output/{job.spec.bubble_file}",
                f"/mg-align-output/{job.metadata.name}.aligned_reads.sam",
                f"/mg-align-output/{job.metadata.name}.output.sam",
            ],
            env=[V1EnvVar(name="APPLICATION", value="sam_pad"),],
        )

        job.status.waiting_for.extend(
            [r.metadata.name for r in api_responses if isinstance(r, V1Job)]
        )

##############################
# k8sbwbble/jobs/__init__.py #
##############################

from .align_job import AlignJob
from .aln2sam_job import Aln2SamJob
from .comb_job import CombJob
```

```
from .dataprep_job import DataPrepJob
from .index_job import IndexJob
from .sampad_job import SampadJob
from .merge_job import MergeJob
from .execution_time_job import ExecutionTimeJob


---

################
# alignjob.yml #
################

---
apiVersion: bwbble.aideen.dev/v1
kind: AlignJob
metadata:
  name: test-job5
  namespace: bwbble-dev
spec:
  alignParallelism: 2
  readsCount: 512000
  readsFile: dummy_reads.fastq
  bubbleFile: chr21_bubble.data
  snpFile: chr21_ref_w_snp_and_bubble.fasta
  bwbbleVersion: "313"
---

#############
# crd.yaml #
#############

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below and be in the form: <plural>.<group>
  name: alignjobs.bwbble.aideen.dev
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: bwbble.aideen.dev
  # list of versions supported by this CustomResourceDefinition
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage version.
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              required: ["readsFile", "readsCount", "bubbleFile", "snpFile"]
              properties:
                readsFile:
                  type: string
```

```
                  description: "The name of the file from which DNA reads will be consumed.
"
                bwbbleVersion:
                  type: string
                  description: "The version of the bwbble toolchain to use."
                  default: "latest"
                readsCount:
                  type: integer
                  description: "The number of reads in the reads file."
                alignParallelism:
                  type: integer
                  description: "The number of containers that will each run a section of th
e reads file."
                  default: 1
                bubbleFile:
                  type: string
                  description: "The reference genome bubble data used in aln2sam to create
aligned reads .SAM file."
                snpFile:
                  type: string
                  description: "The multireference genome file for short-
read alignment of the reads file."
            status:
              type: object
              properties:
                startTime:
                  type: string
                endTime:
                  type: string
                stage:
                  type: string
                waitingFor:
                  type: array
                  items:
                    type: string
                  default: []
                executionTimes:
                  type: object
                  additionalProperties:
                    type: object
                    additionalProperties:
                      type: object
                      required:
                        - total
                      properties:
                        total:
                          type: string
                          description: "The total end-to-
end execution time measured by the Kubernetes API"
                        internal:
                          type: string
                          description: "An internally measured execution time from the appl
ication."

  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
```

```
    plural: alignjobs
    # singular name to be used as an alias on the CLI and for display
    singular: alignjob
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: AlignJob
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
      - aj
      - ajs
---

####################
# deployment.yaml #
####################

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bwbble-controller
  namespace: bwbble-dev
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: bwbble-dev
  name: bwbble-controller
rules:
  - apiGroups: ["bwbble.aideen.dev"]
    resources: ["alignjobs"]
    verbs: ["*"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "watch", "list", "create", "update", "delete"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "create", "update", "delete"]
  - apiGroups: [""]
    resources: ["pods", "pods/log"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bwbble-controller
  namespace: bwbble-dev
subjects:
  - kind: ServiceAccount
    name: bwbble-controller
    namespace: bwbble-dev
roleRef:
  kind: Role
  name: bwbble-controller
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: bwbble-controller
  namespace: bwbble-dev
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bwbble-controller
  template:
    metadata:
      labels:
        app: bwbble-controller
    spec:
      containers:
        - name: controller
          image: bwbble/controller:16
          args:
            - "--namespace=bwbble-dev"

      serviceAccountName: bwbble-controller
```

Appendix 17 BWBBLE Helm Chart

```
---

###################
# aks.values.yaml #
###################

dataprep:
  enabled: false

  ref_genome: chr21.fa
  snp_indel_files:
    - ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf

  # These files are the output of the data preparation phase
  # When dataprep is enabled use ref_w_snp.fasta for snp_file
  snp_file: chr21_ref_w_snp.fasta
  bubble_file: chr21_bubble.data
  snp_bubble_file: chr21_ref_w_snp_and_bubble.fasta

index:
  enabled: true

align:
  enabled: true

  # These files are input for the align phase, generated by the dataprep phase
  # When dataprep is enabled use ref_w_snp.fasta for snp_file
  snp_file: chr21_ref_w_snp.fasta
  bubble_file: chr21_bubble.data

  output: output.sam

  reads:
    file: dummy_reads.fastq
```

```
    ranges:
      - start: 0
        length: -1
      # - start: 128000
      #   length: 128000
      # - start: 256000
      #   length: 128000
      # - start: 384000
      #   length: 128000
      # - start: 50
      #   length: 50

ref:
  vcf:
    file: ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf
  human_genome:
    file: chr21.fa

volumes:
  input:
    azureFile:
      secretName: azure-secret
      shareName: input
      readonly: false
  refoutput:
    azureFile:
      secretName: azure-secret
      shareName: ref-output
      readonly: false
  alignoutput:
    azureFile:
      secretName: azure-secret
      shareName: align-output
      readonly: false

      apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm

---

##########################
# azure-file-secret.yaml #
##########################

apiVersion: v1
data:
  azurestorageaccountkey: # TODO: Fill in with your base64 encoded access key
  azurestorageaccountname: # TODO: Fill in with your base64 encoded storage account name
kind: Secret
metadata:
  name: azure-secret
  namespace: bwbble-dev
```

```
type: Opaque

---

####################
# bwbble.crds.yaml #
####################

apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: bwbble.dataprep
spec:
  version: v1
  group: bwbble
  scope: Namespaced
  names:
    singular: dataprep
    plural: datapreps
    kind: DataPrep
    shortNames:
      - dp
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: bwbble.align
spec:
  version: v1
  group: bwbble
  scope: Namespaced
  names:
    singular: align
    plural: aligns
    kind: Align
    shortNames:
      - al

---

####################
# bwbble.jobs.yaml #
####################

---
apiVersion: bwbble/v1
kind: DataPrep
metadata:
  name: test1
spec:
  ref_genome: human_g1k_v37.fasta
  snp_indel_files:
    - ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf

  # These files are the output of the data preparation phase
  # When dataprep is enabled use ref_w_snp.fasta for snp_file
  snp_file: chr21.fa
  bubble_file: bubble.data
```

```
  snp_bubble_file: ref_w_snp_and_bubble.fasta

  volumes:
    input:
      azureFile:
        secretName: azure-secret
        shareName: input
        readonly: false
    refoutput:
      azureFile:
        secretName: azure-secret
        shareName: ref-output
        readonly: false
    alignoutput:
      azureFile:
        secretName: azure-secret
        shareName: align-output
        readonly: false
---
apiVersion: bwbble/v1
kind: Align
metadata:
  name: test1
spec:
  wait_upon:
    apiVersion: bwbble/v1alpha1
    kind: DataPrepJob
    name: test1

  snp_file: chr21.fa
  bubble_file: bubble.data

  output: output.sam

  reads:
    file: dummy_reads_large.fastq
    parallelism: 8

  volumes:
    input:
      azureFile:
        secretName: azure-secret
        shareName: input
        readonly: false
    refoutput:
      azureFile:
        secretName: azure-secret
        shareName: ref-output
        readonly: false
    alignoutput:
      azureFile:
        secretName: azure-secret
        shareName: align-output
        readonly: false

---
```

```
#######################
# desktop.values.yaml #
#######################

dataprep:
  enabled: false

  ref_genome: chr21.fa
  snp_indel_files:
    - ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf

  # These files are the output of the data preparation phase
  # When dataprep is enabled use ref_w_snp.fasta for snp_file
  snp_file: chr21.fa
  bubble_file: bubble.data
  snp_bubble_file: ref_w_snp_and_bubble.fasta

index:
  enabled: true

align:
  enabled: true

  # These files are input for the align phase, generated by the dataprep phase
  # When dataprep is enabled use ref_w_snp.fasta for snp_file
  snp_file: chr21.fa
  bubble_file: bubble.data

  output: output.sam

  reads:
    file: dummy_reads.fastq
    ranges:
      - start: 0
        length: 50
      - start: 50
        length: 50

ref:
  vcf:
    file: ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf
  human_genome:
    file: human_g1k_v37.fasta

volumes:
  input:
    hostPath:
      path: /host_mnt/c/code/github.com/mang0kitty/bwbble/Docker/input
      type: Directory
  refoutput:
    hostPath:
      path: /host_mnt/c/code/github.com/mang0kitty/bwbble/Docker/ref-output
      type: Directory
  alignoutput:
    hostPath:
      path: /host_mnt/c/code/github.com/mang0kitty/bwbble/Docker/align-output
      #path: /run/desktop/mnt/host/c/code/github.com/mang0kitty/bwbble/Docker/align-output
      type: Directory
```

```
---

#####################
# bwbble/Chart.yaml #
#####################

apiVersion: v2
name: bwbble
description: A Helm chart for running a DNA alignment using BWBBLE on K8s
type: application

# This is the chart version. This version number should be incremented each time you make c
hanges
# to the chart and its templates, including the app version.
version: 0.1.0

# This is the version number of the application being deployed. This version number should
be
# incremented each time you make changes to the application.
appVersion: 1.0.0

---

######################
# bwbble/values.yaml #
######################

# Default values for mg-align-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1

dataprep:
  enabled: true

  image:
    repository: bwbble/mg-ref
    tag: 250

  ref_genome: chr21.fasta
  snp_indel_files:
    - ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf

  # These files are the output of the data preparation phase
  snp_file: ref_w_snp.fasta
  bubble_file: bubble.data
  snp_bubble_file: ref_w_snp_and_bubble.fasta

index:
  enabled: true

align:
  enabled: true
  image:
    repository: bwbble/mg-aligner
    tag: latest
```

```
  # These files are input for the align phase, generated by the dataprep phase
  snp_file: ref_w_snp.fasta
  bubble_file: bubble.data

  output: output.sam

  reads:
    file: sim_chr21_N100.fastq
    ranges:
      - start: 0
        length: 50
      - start: 50
        length: 50

ref:
  image:
    repository: bwbble/mg-ref
    tag: 250
  vcf:
    file: ALL.chr21.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf
  human_genome:
    file: human_g1k_v37.fasta

image:
  pullPolicy: IfNotPresent

nameOverride: ""
fullnameOverride: ""

storageSecrets:
  username: Guest
  password: ""

volumes:
  input: {}
  refoutput: {}
  alignoutput: {}

---

################################
# bwbble/templates/align.yaml #
################################

{{ if .Values.align.enabled }}
{{ range .Values.align.reads.ranges }}
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: "{{ $.Chart.Name }}-align-{{ $.Release.Name }}-range-{{ .start }}-{{ .length }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: align
    bwbble-job: "{{ $.Release.Name }}"
    bwbble-version: "{{ $.Chart.AppVersion }}"
data:
```

```
  container_args: "align -s {{ .start }} -p {{ .length }} '/mg-ref-
output/{{ $.Values.dataprep.snp_file }}' '/input/{{ $.Values.align.reads.file }}' '/mg-
align-output/{{ $.Release.Name }}.aligned_reads.{{ .start }}.aln'"
---
# align for range {{ .start }}..{{ .length }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ $.Chart.Name }}-align-{{ $.Release.Name }}-range-{{ .start }}-{{ .length }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: align
    bwbble-job: "{{ $.Release.Name }}"
    bwbble-version: "{{ $.Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: align
      labels:
        app.kubernetes.io/managed-by: faaideen
        bwbble-stage: align
        bwbble-job: "{{ $.Release.Name }}"
    spec:
      restartPolicy: Never
      {{ if or ($.Values.dataprep.enabled) ($.Values.index.enabled) }}
      initContainers:
      - name: wait-for-index-job
        image: bitnami/kubectl
        args:
        - "wait"
        - "--for=condition=complete"
        - "job/{{ $.Chart.Name }}-index-{{ $.Release.Name }}"
        - "--timeout=-1s"
      {{ end }}
      containers:
        - name: align
          image: "{{ $.Values.align.image.repository }}:{{ $.Values.align.image.tag }}"
          imagePullPolicy: "{{ $.Values.image.pullPolicy }}"
          env:
            - name: ARGS_FILE
              value: /var/run/args/container_args
          resources:
            requests:
              memory: "1.5Gi"
              cpu: 0.5
            limits:
              memory: "4Gi"
              cpu: 1
          volumeMounts:
            - mountPath: /var/run/args
              name: args
            - mountPath: /input
              name: input
            - mountPath: /mg-ref-output
              name: ref-output
            - mountPath: /mg-align-output
              name: align-output
```

```
      volumes:
        - name: args
          configMap:
            name: "{{ $.Chart.Name }}-align-{{ $.Release.Name }}-range-{{ .start }}-
{{ .length }}"
        - name: input
{{ toYaml $.Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml $.Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml $.Values.volumes.alignoutput | indent 10 }}
{{ end }}
{{ end }}
---

################################
# bwbble/templates/aln2sam.yaml #
################################

{{ if .Values.align.enabled }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-aln2sam-{{ .Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: aln2sam
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: aln2sam
      labels:
        app.kubernetes.io/managed-by: faaideen
    spec:
      restartPolicy: Never
      initContainers:
      - name: wait-for-merge-job
        image: bitnami/kubectl
        args:
        - "wait"
        - "--for=condition=complete"
        - "job/{{ .Chart.Name }}-merge-{{ .Release.Name }}"
        - "--timeout=-1s"
      containers:
        - name: aln2sam
          image: "{{ .Values.align.image.repository }}:{{ .Values.align.image.tag }}"
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          args:
            - aln2sam
            - "/mg-ref-output/{{ .Values.align.snp_file }}"
            - "/input/{{ .Values.align.reads.file }}"
            - "/mg-align-output/{{ .Release.Name }}.aligned_reads.aln"
            - "/mg-align-output/{{ .Release.Name }}.aligned_reads.sam"
          volumeMounts:
            - mountPath: /input
              name: input
```

```
            - mountPath: /mg-ref-output
              name: ref-output
            - mountPath: /mg-align-output
              name: align-output
      volumes:
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 10 }}
{{ end }}
---

##############################
# bwbble/templates/comb.yaml #
##############################

{{ if .Values.dataprep.enabled }}
apiVersion: v1
kind: ConfigMap
metadata:
  name: "{{ .Chart.Name }}-comb-{{.Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: comb
    bwbble-job: "{{ $.Release.Name }}"
    bwbble-version: "{{ $.Chart.AppVersion }}"
data:
  container_args: "'/input/{{ .Values.dataprep.ref_genome }}' '/mg-ref-
output/{{ .Values.dataprep.snp_file }}' '/mg-ref-
output/{{ .Values.dataprep.snp_bubble_file }}' '/mg-ref-
output/{{ .Values.dataprep.bubble_file }}'"
---
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-comb-{{.Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: comb
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: comb
      labels:
        app.kubernetes.io/managed-by: faaideen
    spec:
      restartPolicy: Never
      initContainers:
      - name: wait-for-dataprep-job
        image: bitnami/kubectl
        args:
        - "wait"
        - "--for=condition=complete"
        - "job/{{ .Chart.Name }}-dataprep-{{ .Release.Name }}"
```

```
        - "--timeout=-1s"
      containers:
        - name: comb
          image: "{{ .Values.ref.image.repository }}:{{ .Values.ref.image.tag }}"
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          env:
            - name: APPLICATION
              value: comb
            - name: ARGS_FILE
              value: /var/run/args/container_args
          resources:
            requests:
              memory: "8Gi"
              cpu: 1
            limits:
              memory: "32Gi"
              cpu: 4
          volumeMounts:
            - mountPath: /var/run/args
              name: args
            - mountPath: /input
              name: input
            - mountPath: /mg-ref-output
              name: ref-output
            - mountPath: /mg-align-output
              name: align-output
      volumes:
        - name: args
          configMap:
            name: "{{ .Chart.Name }}-comb-{{.Release.Name }}"
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 10 }}
{{ end }}
---

##################################
# bwbble/templates/dataprep.yaml #
##################################

{{ if .Values.dataprep.enabled }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-dataprep-{{.Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: dataprep
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: dataprep
      labels:
```

```
        app.kubernetes.io/managed-by: faaideen
        bwbble-stage: dataprep
        bwbble-version: "{{ .Chart.AppVersion }}"
    spec:
      restartPolicy: Never
      containers:
        - name: dataprep
          image: "{{ .Values.ref.image.repository }}:{{ .Values.ref.image.tag }}"
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          args:
            - data_prep
            {{ range .Values.dataprep.snp_indel_files }}
            - "/input/{{ . }}"
            {{ end }}
          volumeMounts:
            - mountPath: /input
              name: input
            - mountPath: /mg-ref-output
              name: ref-output
            - mountPath: /mg-align-output
              name: align-output
      volumes:
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 10 }}
{{ end }}
---

########################################
# bwbble/templates/decompress_vcf.yaml #
########################################

apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-decompress-{{ .Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: decompress
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: decompress
      labels:
        app.kubernetes.io/managed-by: faaideen
    spec:
      restartPolicy: Never
      containers:
        - name: decompress
          image: ubuntu:latest
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          args:
            - bash
```

```
            - "-c"
            - "cd /input && gzip -d /input/{{ index .Values.dataprep.snp_indel_files 0
}}.gz"
          volumeMounts:
            - mountPath: /input
              name: input
      volumes:
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}

---

################################
# bwbble/templates/index.yaml #
################################

{{ if or ($.Values.dataprep.enabled) ($.Values.index.enabled) }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-index-{{ .Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: index
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: index
      labels:
        app.kubernetes.io/managed-by: faaideen
        app.kubernetes.io/managed-by: faaideen
        bwbble-stage: index
        bwbble-version: "{{ .Chart.AppVersion }}"
    spec:
      restartPolicy: Never
      {{ if $.Values.dataprep.enabled }}
      initContainers:
      - name: wait-for-comb-job
        image: bitnami/kubectl
        args:
        - "wait"
        - "--for=condition=complete"
        - "job/{{ .Chart.Name }}-comb-{{ .Release.Name }}"
        - "--timeout=-1s"
      {{ end }}
      containers:
        - name: index
          image: "{{ .Values.align.image.repository }}:{{ .Values.align.image.tag }}"
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          args:
            - index
            - "/mg-ref-output/{{ .Values.dataprep.snp_file }}"
          resources:
            requests:
              memory: "50Gi"
              cpu: 8
```

```
            limits:
              memory: "64Gi"
              cpu: 8
          volumeMounts:
            - mountPath: /input
              name: input
            - mountPath: /mg-ref-output
              name: ref-output
            - mountPath: /mg-align-output
              name: align-output
          resources:
            requests:
              memory: "40Gi"
              cpu: 1
            limits:
              memory: "64Gi"
              cpu: 4
      volumes:
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 10 }}
{{ end }}
---

################################
# bwbble\templates\merge.yaml #
################################

{{ if .Values.align.enabled }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-merge-{{ .Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: merge
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: merge
      labels:
        app.kubernetes.io/managed-by: faaideen
    spec:
      restartPolicy: Never
      initContainers:
      - name: wait-for-align-job
        image: bitnami/kubectl
        args:
        - "wait"
        - "--for=condition=complete"
        - "job"
        - "-l"
        - "bwbble-job={{ .Release.Name }}"
```

```
        - "-l"
        - "bwbble-stage=align"
        - "--timeout=-1s"
      containers:
        - name: merge
          image: busybox
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          args:
            - "sh"
            - "-c"
            - "cat {{ range .Values.align.reads.ranges }} /mg-align-
output/{{ $.Release.Name }}.aligned_reads.{{ .start }}.aln {{ end }} > /mg-align-
output/{{ .Release.Name }}.aligned_reads.aln"
          volumeMounts:
          - mountPath: /input
            name: input
          - mountPath: /mg-ref-output
            name: ref-output
          - mountPath: /mg-align-output
            name: align-output
      volumes:
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 10 }}
{{ end }}
---

##################################
# bwbble/templates/sam_pad.yaml #
##################################

{{ if .Values.dataprep.enabled }}
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Chart.Name }}-sampad-{{.Release.Name }}"
  labels:
    app.kubernetes.io/managed-by: faaideen
    bwbble-stage: sam_pad
    bwbble-version: "{{ .Chart.AppVersion }}"
spec:
  completions: 1
  template:
    metadata:
      name: sampad
      labels:
        app.kubernetes.io/managed-by: faaideen
    spec:
      restartPolicy: Never
      initContainers:
      - name: wait-for-aln2sam-job
        image: bitnami/kubectl
        args:
        - "wait"
        - "--for=condition=complete"
```

```
        - "job/{{ .Chart.Name }}-aln2sam-{{ .Release.Name }}"
        - "--timeout=-1s"
      containers:
        - name: sampad
          image: "{{ .Values.ref.image.repository }}:{{ .Values.ref.image.tag }}"
          imagePullPolicy: "{{ .Values.image.pullPolicy }}"
          args:
            - sam_pad
            - "/mg-ref-output/{{ .Values.align.bubble_file }}"
            - "/mg-align-output/{{ .Release.Name }}.aligned_reads.sam"
            - "/mg-align-output/{{ .Values.align.output }}"
          volumeMounts:
            - mountPath: /input
              name: input
            - mountPath: /mg-ref-output
              name: ref-output
            - mountPath: /mg-align-output
              name: align-output
      volumes:
        - name: input
{{ toYaml .Values.volumes.input | indent 10 }}
        - name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 10 }}
        - name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 10 }}
{{ end }}
---

##################################
# bwbble/templates/_helpers.tpl #
##################################

{{/* vim: set filetype=mustache: */}}
{{/*
Expand the name of the chart.
*/}}
{{- define "mg-align-chart.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}

{{/*
Create a default fully qualified app name.
We truncate at 63 chars because some Kubernetes name fields are limited to this (by the DNS
 naming spec).
If release name contains chart name it will be used as a full name.
*/}}
{{- define "mg-align-chart.fullname" -}}
{{- if .Values.fullnameOverride -}}
{{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" -}}
{{- else -}}
{{- $name := default .Chart.Name .Values.nameOverride -}}
{{- if contains $name .Release.Name -}}
{{- .Release.Name | trunc 63 | trimSuffix "-" -}}
{{- else -}}
{{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
{{- end -}}
{{- end -}}
{{- end -}}
```

```
{{/*
Create chart name and version as used by the chart label.
*/}}
{{- define "mg-align-chart.chart" -}}
{{- printf "%s-%s" .Chart.Name .Chart.Version | replace "+" "_" | trunc 63 | trimSuffix "-
" -}}
{{- end -}}

{{/*
Common labels
*/}}
{{- define "mg-align-chart.labels" -}}
app.kubernetes.io/name: {{ include "mg-align-chart.name" . }}
helm.sh/chart: {{ include "mg-align-chart.chart" . }}
app.kubernetes.io/instance: {{ .Release.Name }}
{{- if .Chart.AppVersion }}
app.kubernetes.io/version: {{ .Chart.AppVersion | quote }}
{{- end }}
app.kubernetes.io/managed-by: {{ .Release.Service }}
{{- end -}}

{{ define "defaultVolumeMounts" -}}
- mountPath: /input
  name: input
- mountPath: /mg-ref-output
  name: ref-output
- mountPath: /mg-align-output
  name: align-output
{{- end }}

{{ define "defaultVolumes" -}}
- name: input
{{ toYaml .Values.volumes.input | indent 2 }}
- name: ref-output
{{ toYaml .Values.volumes.refoutput | indent 2 }}
- name: align-output
{{ toYaml .Values.volumes.alignoutput | indent 2 }}
{{- end }}
```