CurateND

UNIVERSITY OF NOTRE DAME | HESBURGH LIBRARIES

Expediting Analysis and Improving Fidelity of Big Data Genomics

Olivia Choudhury

Publication Date

07-07-2017

License

This work is made available under a All Rights Reserved license and should only be used in accordance with that license.

Citation for this work (American Psychological Association 7th edition)

Choudhury, O. (2017). *Expediting Analysis and Improving Fidelity of Big Data Genomics* (Version 1). University of Notre Dame. https://doi.org/10.7274/zp38w953f1h

This work was downloaded from CurateND, the University of Notre Dame's institutional repository.

For more information about this work, to report or an issue, or to preserve and share your original work, please contact the CurateND team for assistance at curate@nd.edu.

EXPEDITING ANALYSIS AND IMPROVING FIDELITY OF BIG DATA GENOMICS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Olivia Choudhury

Scott J. Emrich, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana

July 2017

© Copyright by

Olivia Choudhury

2017

All Rights Reserved

EXPEDITING ANALYSIS AND IMPROVING FIDELITY OF BIG DATA GENOMICS

Abstract

by

Olivia Choudhury

Genomics, or the study of genome-derived data, has had widespread impact in applications including medicine, forensic science, human evolution, environmental science, and social science. The plummeting cost of genome sequencing in the last decade has spurred an exponential growth of genomic data. The rate of data generation from these sequencing techniques has outpaced computing throughput, as predicted by Moore's Law, causing a major bottleneck in the rate of data processing and analysis. Emerging genome data is also characterized by missing and erroneous values, that reduce data fidelity and limit its applicability for downstream analysis. This forms the basis of the following research questions: (i) Can we design frameworks that can expedite data analysis and enable efficient utilization of computational resources? (ii) Can we develop accurate and efficient algorithms to improve data fidelity in genomic applications?

We address the first problem by developing a parallel data analysis framework that accelerates large-scale comparative genomics applications. We identify that optimal data partitioning and caching significantly improve the performance of such framework. We further construct a predictive model to estimate runtime configurations that facilitate optimal utilization of cloud and cluster-based resources while executing data-intensive applications.

The fidelity of genomic data derived from next-generation sequencing techniques impacts downstream applications like genome-wide association study (GWAS) and genome assembly. For imputation of missing genotype data, we design an accurate, fast, and lightweight algorithm for both model (with a reference genotype panel) and non-model (without a reference genotype panel) organisms. To correct erroneous long reads generated by emerging sequencing techniques, we formulate a hybrid correction algorithm that determines a correction policy based on an optimal combination of base quality and similarity of aligned short reads. We extend the core algorithm by proposing an iterative learning paradigm that further improves its performance.

Our proposed data analysis framework is accessible to the scientific community and has been used to study the genomes of important plant species and malaria vector mosquitoes. The predictive models exhibit high accuracy in determining optimal parameters of operation on commercial cloud services like Amazon EC2 and Microsoft Azure. Finally, the imputation and error correction algorithms outperform state-of-the-art alternatives when tested on real data sets of plants, malarial mosquitoes, and humans. Hence, in this thesis, we present novel solutions to expedite data-parallel genomic applications while optimizing cloud and cluster-based resource utilization. We also design novel, accurate, and efficient algorithms to impute missing data and correct erroneous data in emerging genomic applications.

To my grandmother, Amita Mukherjee (1924 - 2001).

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

Genes don't think about what constitutes good or evil. They don't care whether we are happy or unhappy. We're just means to an end for them. The only thing they think about is what is most efficient for them.

— Haruki Murakami, 1Q84

I would like to thank Dr. Scott Emrich for his support and guidance as my PhD advisor. He has helped me grow as an independent researcher, and present my research in an articulate manner. I am grateful to my committee members Dr. Kevin Bowyer, Dr. Jeanne Romero-Severson, and Dr. Douglas Thain for the invaluable feedback that helped shape this thesis. I am deeply indebted to the Eck Institute for Global Health (EIGH) for their generous support that enabled me to disseminate my research and improve my work through wonderful interactions at various academic venues.

I have been lucky to have had brilliant labmates like Dr. Lauren Assour, Dr. Irena Lanc, Aaron Steele, Dr. Andrew Thrasher, Dr. Wei Zhang, Colin Teberg, Shenglong Zhu, and Gabe Wright. I have had a great experience working with our collaborators in the Department of Biology at University of Notre Dame: Dr. Arpita Konar, Dr. Rachel Love, Rachel Wiltshire, and Dr. Candice Lumibao. You have all helped me grasp the biological underpinning of the research problems and the underlying nuances. I am also thankful to Nicholas Hazekamp, Dr. Sandra Gesing, Dr. Joseph Sarro, Dr. Benjamin Tovar, Dr. Haiyan Meng, Dr. Dinesh Rajan, and Dr. Li Yu for helpful conversations and collaborations.

I owe my sincere gratitude to William Stern and Scott Symes, with whom I had the pleasure of working during my internship at IBM Watson in summer 2015. I feel fortunate for getting the opportunity to work at the Broad Institute of MIT and Harvard with Dr. Daniel Neafsey, Dr. Seth Redmond, Dr. Aimee Taylor, and Dr. Angela Early. I would also like to thank Dr. Rajat K. De at the Indian Statistical Institute, Kolkata, for introducing me to the field of bioinformatics.

I want to thank Graduate Society of Women Engineers (GradSWE) and Graduate Student Union (GSU) at University of Notre Dame for giving me an opportunity to serve for the graduate student community.

A hearty thank you to all my friends. To Tariq Iqbal, Dr. Fazle Faisal, Robert Perricone, Salvador Aguinaga, Maryam Moosaei, and John Bernhard for the group study sessions. To Dr. Michael Gonzales, Dr. Cory Hayes, Dr. Vipin Vijayan, Siyuan Jiang, and Mark Costanzo for the game nights. To Valentina Geri, Katherine Ramos, Nunzia Pirro, and Valentina Damioli for the relaxing evenings and rudimentary Spanish/Italian. To Shinjini Chattopadhyay and Krishna Namboothiri for pampering me with good food. To Meenakshi Chatterjee, Shanika Hapuarachchi, and Dilshan Godaliyadda for the impromptu holiday plans. To Shriya Banerjee and Supratim Chaudhury for many intriguing conversations. To Dr. Dipanwita Dasgupta, Rusha Chatterjee, and Aastha Nigam for always being there for me.

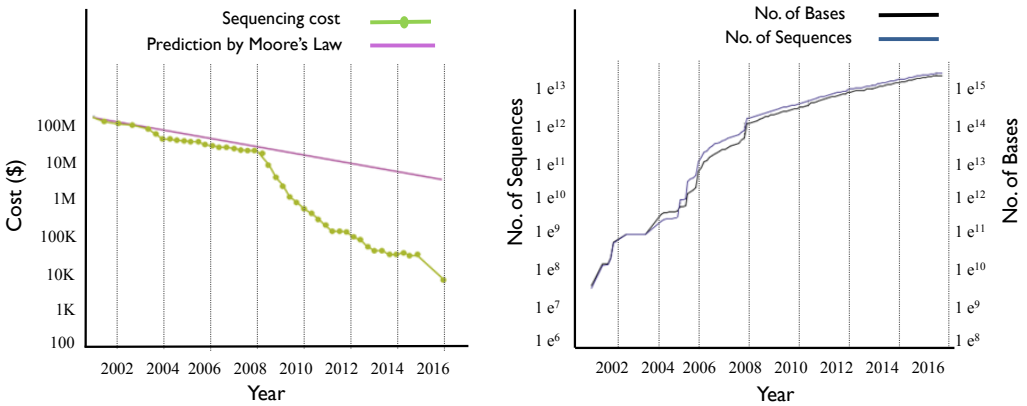
A special thank you to Dr. Ankush Chakrabarty for being the best friend and mentor one could ask for. Most importantly, I would like to thank my parents: Anand Choudhury and Bhaswati Choudhury for encouraging me to pursue my dreams, instilling the value of sincerity and honesty, and setting an exceptional example to grow as an individual.

CHAPTER 1

INTRODUCTION

1.1 Motivation And Problem Statement

Genome data analysis is the key to deciphering research problems in the areas of medicine [69, 108], evolution [24, 59], environmental science [48, 128], forensic studies [62, 71], and social science [44, 50].



(A) Reduced Sequencing Cost

(B) Genomic Data Growth

Figure 1.1: Reduction of genome sequencing cost and inflation of genomic data in the last decade. (A) Real cost (green dotted line) of sequening a genome and estimated cost (pink line), based on Moore's law. (B) Number of genome sequences (blue line) and bases (black line) produced by sequencing platforms.

Next-generation sequencing (NGS) techniques have revolutionalized the field of genomics by generating genome data at an unprecedented rate, thereby facilitating previously inconceivable scientific discoveries [104, 125]. However, the rate of data generation has outpaced computing throughput, as predicted by Moore's Law [113], causing a major hurdle in efficient analysis of large-scale genome data (Figure 1.1).

Furthermore, data generated from emerging sequencing platforms are often rife with missing and specious values, causing a significant decline in its fidelity and applicability for subsequent analysis. For instance, quality of inferred knowledge obtained from genome-wide association study (association of mutations with traits, such as disease susceptibility) is undermined due to missing genotypes in the input data. *De novo* genome assembly, another downstream application, produce low-quality assemblies due to the use of error-prone data derived from PacBio Sequencing [45] and Oxford Nanopore Sequencing [27]. Hence, expediting analysis and improving fidelity of large-scale genome data are imperative in advancing the pace and quality of innovations in the domain of genomics.

- *The first research problem investigated in this thesis is to develop efficient data analysis frameworks to expedite big data genomics and predictive models to optimize its required computational resources.*
- *The second problem investigated in this thesis is to design accurate and efficient imputation and error correction algorithms to improve fidelity of genome data.*

According to the classical definition of "big data", the above-mentioned research problems closely relate to addressing the challenges associated with big data analysis: volume, velocity, and veracity. The solutions presented in this thesis can be potentially adapted in other fields of big data analysis.

1.2 Literature Review

1.2.1 Data Analysis Framework For Expediting Comparative Genomics Applications

Genome mapping and assembly, the two important applications in comparative genomics, have traditionally been refactored with Message Passing Interface (MPI) [39] and MapReduce frameworks [40]. ClustalW-MPI [94] adopts MPI to build a scalable algorithm for multiple sequence alignment. CloudBurst [123] is one of the early applications that implements the seed-and-extend genome mapping algorithm on a Hadoop-based framework [4]. When tested on billions of reads, typical of current large-scale genomics projects, it incurs higher computation time and resources than other cloud-based variant detection tools like Crossbow [82]. However, the underlying method of Crossbow does not support some desired features, such as allowing gaps in sequence alignment. SEAL [117], another genome alignment tool, uses MapReduce for parallelization of tasks.

Although MPI enables execution of tasks in parallel, it involves complex software development for refactoring tools [120]. Hadoop's MapReduce-based framework does not readily allow tuning of runtime parameters, a requirement of many applications. Also, dynamic scaling of resources based on workload is a tedious process in Hadoop due to the tight coupling between compute runtime and data storage layers [101, 141]. Finally, its performance degrades when computational resources are harnessed from a heterogeneous system [145].

We develop an efficient data analysis framework that supports the above-mentioned features by leveraging the Makeflow language [144] and Work Queue master-worker framework [20]. We propose various strategies of data partitioning to determine an optimal scheme that reduces computational overhead incurred by large-scale workloads. We explore different methods of workflow fusion that further expedites data

analysis. Finally, we illustrate the application of this framework in analyzing multiple real data sets, particularly in constructing a high-quality genetic map of the northern red oak tree, the first of its kind.

1.2.2 Computational Resource Optimization For Executing Data-Intensive Genomic Applications On Clusters And Clouds

Analysis of large-scale data, particularly in areas like bioinformatics [139], high energy physics [47], and biometrics [114], demands high computational time and resources. Scientists often resort to commercial clouds or clusters to process such data. Among the existing infrastructures, Hadoop-based methods such as G-Hadoop [140] allows efficient data analysis using multiple clusters that are distributed across data centers. Dryad [67] provides a programming framework and distributed execution engine for directed acyclic graph (DAG)-based workloads. CIEL [115] extends its programming and execution models to support dynamic data dependencies and arbitrary data-dependent control flows. Scheduling techniques reported in [23, 53, 118] reduce computational burden of large workloads in a distributed system.

As noted in [70], predictive models play a vital role in efficient management and operation of workloads in a distributed system. The authors in [66], [54] also highlight its importance in optimizing computational resource usage. However, none of the above-mentioned frameworks support a predictive model to estimate an optimal runtime configuration that can reduce resource utilization in a distributed environment. We develop a hybrid model that predicts optimal runtime configuration (number of threads, number of tasks, etc.) for a given class of applications to minimize cluster and cloud-based resource usage. Although earlier studies [8, 43, 106] have explored the applicability of machine learning algorithms in estimating resource utilization, our proposed model supports applications that allow multithreading as well as distributed computing. We reduce resource utilization and cost of operation by

finding the trade-off between parallelism achieved from multithreading (thread-level parallelism) and distributed computing (task-level parallelism).

1.2.3 Imputation Of Missing Genotype Data In Model And Non-Model Organisms

Early work on genotype imputation leveraged the principle of Expectation Maximization (EM) [41], which is computationally prohibitive for large-scale data sets. More recent tools such as IMPUTE2 [64], generate approximate coalescent models and hidden Markov models (HMMs) from genotypes for subsequent stochastic EM-based algorithms. PHASE [130] uses Markov chain Monte-Carlo (MCMC) algorithm to explore possible combinations of haplotypes [95]. The combinatorial explosion inherent in MCMC limits its applicability to small datasets [18]. FastPHASE [124], a faster variant of PHASE, implements a parsimonious clustering of haplotypes and is more amenable to medium-sized samples. For large datasets, this algorithm uses a subset of haplotypes, resulting in performance degradation. To overcome this challenge in large samples, Beagle [16] employs HMM to find haplotype clustering at individual loci.

Majority of the leading imputation tools, such as IMPUTE2, PHASE, and Beagle, rely on reference genotype panels and a genetic map, thereby limiting itself to only model organisms. Due to the availability of reference panels, machine learning-based methods like artificial neural networks have been used to solve similar problems in model organisms [131]. For non-model organisms (without reference panels), LinkImpute [112], based on a variant of k-nearest neighbor interpolation, is currently the only imputation tool available. However, it is sluggish and exhibits limited accuracy when used in large data sets.

ADDIT, our proposed imputation tool, is the first of its kind to be applicable to both model and non-model organisms. For the former, its imputation policy is based on data-driven learning algorithm that considers cases with more than two putative

values or alleles, as opposed to the existing bi-classification methods. For non-model organisms, it provides a fast, accurate, and lightweight approach of imputation without requiring additional data sources.

1.2.4 Hybrid Correction Of Erroneous Long Reads Using Iterative Learning

Existing error correction methods for long reads can be categorized as self-correction or hybrid correction algorithm. HGAP [29], a self-correction tool, selects the longest reads as seeds and aligns all other reads to them. During a preassembly stage, the seed reads are converted to more accurate reads that can then be used for downstream analysis. As this approach solely relies on long reads for correction, it requires the reads to have very high coverage, typically on the order of 50X. Given the higher cost of long read sequencing, such high coverage may not always be practical.

Hybrid error correction tools like LSC [14], PacBioToCA [76], proovread [56], LoRDEC [121], and CoLoRMap [57] leverage alignment information from accurate short reads sequenced from the same or highly similar individuals. LSC and PacBioToCA largely follow a consensus-based (majority vote) correction approach. proovread also exploits aligned information from short reads for correction, although it uses iterative mapping to achieve higher senstitivity. LoRDEC constructs a de Bruijn graph from k-mers of the short reads. For an erroneous region on the long read, it finds an optimal overlapping path in the de Bruijn graph to determine a corrected long read. CoLoRMap aligns short and long reads to build an overlap graph. The shortest path in the graph that minimizes edit distance between the aligned regions is then used for correction. Nanocorr, the error correction tool for Oxford Nanopore reads, deploys a dynamic programming approach for computing the longest-increasing-subsequences (LIS) and overlaps to find a consensus sequence.

Majority voting and optimal path-based approaches, however, may not generate an optimal solution for every erroneous base, especially when more localized data

(e.g., quality and possible variant information between different individuals) from the alignments are available. The authors in [73] have emphasized the importance of incorporating quality values while correcting noisy sequence data. HECIL, our proposed hybrid error correction algorithm, incorporates such localized information while determining the correction policy. Also, to the best of our knowledge, this is the first time an iterative learning paradigm has been used wherein low-confidence corrections (determined by less reliable alignments during processing) can be further investigated and updated using previously corrected regions of the long reads.

1.3 Contributions In This Thesis

The contributions of this thesis include:

- developing a framework for efficient analysis of data-intensive comparative genomics applications;
- designing various strategies of data partitioning and workflow fusion to expedite data analysis on a distributed system;
- building a highly accurate predictive model that determines optimal runtime configuration for large-scale applications supporting multithreading and distributed computing;
- constructing ADDIT, the first novel algorithm for imputing missing genotypes in both model and non-model organisms;
- illustrating superior performance of ADDIT against state-of-the-art tools on real data sets, including humans;
- devising HECIL, a novel hybrid error correction algorithm that introduces an iterative learning approach;
- demonstrating superior performance of HECIL against state-of-the-art techniques in real data sets, including important malaria vector mosquito *Anopheles funestus*.

1.4 Organization Of This Thesis

The rest of the thesis is organized as follows.

In Chapter 2, we present the design and implementation of a parallel framework for comparative genomics applications [33]. We propose various methods of data partitioning and workflow fusion to reduce the computational overhead incurred while processing massive genomic data on a distributed system. We validate the efficacy of our framework on two most important comparative genomics applications: genome mapping and variant detection. When tested on real data sets, our framework reduced the computation time from 12 days to under 2 hours.

In Chapter 3, we build a hybrid model to predict optimal runtime configurations for executing large workloads on clusters and clouds [34]. It constructs an application-specific model to estimate runtime and memory usage, which is then embedded in a more generic system-level model. We implement regression-based methods to design its two-tiered structure. We illustrate predictive capability of the model with respect to runtime, speedup, and cost of operation. We show that for applications enabling both multithreading and distributed computing, it is crucial to balance thread-level and task-level parallelisms, that is, find a sweet spot. This ensures efficient usage of computational resources while processing massive data on clusters and clouds.

In Chapter 4, we devise ADDIT [35, 36], a highly accurate, fast, and lightweight algorithm to impute missing genotype data in model (with a reference data set) and non-model (without a reference data set) organisms. For model organisms, we implement a multi-class supervised learning method to extricate relevant information from reference data panels that enhances imputation accuracy. However, due to the lack of reference data in non-model organisms, we design a novel data-driven technique of imputation wherein adaptive windows and trust metrics are utilized to incorporate underlying local and global information. We demonstrate that for real data sets of humans and plants, ADDIT consistently outperforms state-of-the-art techniques in terms of imputation accuracy, runtime, and memory usage.

In Chapter 5, we develop HECIL, a novel hybrid algorithm for correction of er-

roneous long reads. We align highly accurate short reads, generated from the same individuals or colony, to the error-prone long reads and determine positions of disagreement or error. We assign a trust metric to evaluate reliability of the underlying short reads before leveraging them for correction. We introduce an iterative learning approach, wherei low-confidence corrections are further investigated and corrected based on updated context of partially corrected long reads obtained from previous iterations. We measure the efficacy of HECIL with respect to k-mer-based, alignment-based, and assembly-based evaluation metrics. A comparative study with state-of-the-art hybrid correction algorithms on real data sets of *E. coli*, *S. cerevisiae*, and the malaria vector mosquito *A. funestus*, reveals that HECIL generates more accurate long reads for higher quality downstream applications like *de novo* assembly.

In Chapter 6, we summarize the contributions of this research and discuss the open problems arising from it.

1.5 Publications

The following publications resulted from this thesis:

- O. Choudhury, A. Chakrabarty, S. Emrich. *A Hybrid Error Correction Algorithm for Long Reads with Iterative Learning.* Submitted, Bioinformatics, 2017.
- O. Choudhury, A. Chakrabarty, S. J. Emrich. *Highly accurate and effcient data-driven methods for genotype imputation.* IEEE/ACM Transactions on Computational Biology and Bioinformatics, PP(99):1-1, 2017. ISSN 1545-5963. doi: 10.1109/TCBB.2017.2708701.
- A. Konar, O. Choudhury, T. McCleary, S. Schlarbaum, O. Gailing, M. Coggeshall, S. Emrich, M. Staton, M. Pfrender, J. Carlson, J. Romero-Severson. *High-quality genetic mapping with ddRADseq in the non-model tree Quercus rubra.* Accepted, BMC Genomics, 2017.
- N. Hazekamp, N. Kremer-Herman, B. Tovar, H. Meng, O. Choudhury, S. Emrich, D. Thain. *Combining Static and Dynamic Storage Management for Data Intensive Scientific Workflows.* Accepted with revision, IEEE Transactions on Parallel and Distributed Systems, 2017.

- O. Choudhury, A. Chakrabarty, S. Emrich. *HAPI-Gen: Highly Accurate Phasing and Imputation of Genotype Data*. ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, 2016.

- A. Konar, O. Choudhury, O. Gailing, M. Coggeshall, M. Staton, S. Emrich, J. Carlson, J. Romero-Severson. *A Genetic Map for the Lobatae*. lnternational Oaks, No. 27, 2016.

- O. Choudhury, N. Hazekamp, D. Thain, S. Emrich. *Accelerating Comparative Genomics Workflows in a Distributed Environment with Optimized Data Partitioning and Workflow Fusion*. Scalable Computing: Practice and Experience 16(1), 2015.

- O. Choudhury, D. Rajan, N. Hazekamp, S. Gesing, D. Thain, S. Emrich. *Balancing Thread-level and Task-level Parallelism for Data-Intensive Workloads on Clusters and Clouds*. IEEE International Conference on Cluster Computing, 2015.

- N. Hazekamp, J. Sarro, O. Choudhury, S. Gesing, S. Emrich, D. Thain. *Scaling Up Bioinformatics Workflows with Dynamic Job Expansion: A Case Study Using Galaxy and Makeflow*. IEEE International Conference on eScience, 2015.

- O. Choudhury, N. Hazekamp, D. Thain, S. Emrich. *Accelerating Comparative Genomic Workflows in a Distributed Environment with Optimized Data Partitioning*. C4Bio at IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2014.

1.6 Availability Of Software

- The data analysis framework described in Chapter 2 is available on Github:
  `https://github.com/cooperative-computing-lab/cctools/tree/master/apps/makeflow_bwa`
  `https://github.com/cooperative-computing-lab/cctools/tree/master/apps/makeflow_gatk`

- The source code of ADDIT described in Chapter 4 is available on Github:
  `https://github.com/NDBL/ADDIT`

- The source code of HECIL described in Chapter 5 is available on Github:
  `https://github.com/NDBL/HECIL`

CHAPTER 2

DATA ANALYSIS FRAMEWORK FOR EXPEDITING COMPARATIVE GENOMICS APPLICATIONS

The following manuscripts describe the work in this chapter.

- O. Choudhury, N. Hazekamp, D. Thain, S. Emrich. *Accelerating Comparative Genomics Workfows in a Distributed Environment with Optimized Data Partitioning and Workfow Fusion.* Scalable Computing: Practice and Experience 16(1), 2015.
- O. Choudhury, N. Hazekamp, D. Thain, S. Emrich. *Accelerating Comparative Genomic Workfows in a Distributed Environment with Optimized Data Partitioning.* C4Bio at IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2014.

2.1 Background

Next-generation sequencing (NGS) techniques have had widespread impact in fields such as molecular medicine, evolution, human migration, DNA forensics, and agriculture [119]. They generate high-throughput DNA fragments, called 'reads', that are crucial in several downstream analysis pipelines [109]. The surge in the rate of data generation demands high computational resources for subsequent analysis. Due to the disparity in sequencing throughput and computational capability, the major research bottleneck lies in the speed of processing such massive biological data. For instance, genome alignment and variant detection, the two most important applications in comparative genomics, often require weeks or months to process large-scale NGS data. A potential approach to reduce computational overhead is to design efficient algorithms or data structures that can prune the search space of operation [91].

However, due to the massive size of NGS data and the high computational resource required for its analysis, designing a parallel data analysis framework that can harness resources from distributed systems is a more tractable solution.

In the last few years, many alignment and variant discovery tools have implemented sophisticated algorithms to reduce computational resource requirement and runtime for analyzing massive genome data [74, 80, 83, 93, 97, 99]. The extent of parallelism that can be achieved from multithreading is often limited. For instance, Burrows Wheeler Aligner (BWA) [88], one of the fastest and most accurate alignment tools currently available, was only able to cause a 5x improvement in speedup for analyzing our test data set (50 oak samples) using multithreading. Similarly, the popular variant detection tool from GATK's HaplotypeCaller [42] required 12 days to determine variants in our test data.

As the underlying algorithms of both tools support coarse-grained parallelism, the trade-off between accuracy and speedup can be mitigated by creating a parallel framework or workflow [79]. Such a workflow can harness resources from distributed systems by dividing the workload into multiple independent tasks and executing the tasks in parallel. In this regard, one of the key factors in achieving a considerable speedup is efficient partitioning of data at the preliminary stage of the workflow. We also identify that merging independent yet interrelated workflows can benefit from efficient caching of data, thereby further improving performance. Hence, we explore different strategies of data partitioning and workflow fusion to determine the optimal design of our framework.

2.2 Methods

2.2.1 Overview Of Genome Alignment And BWA

Millions of short reads generated by NGS techniques are compared to the genome of a model organism, known as the reference genome, for downstream analysis. Genome alignment is the method of comparing query sequences to a reference genome to determine the level of similarity between the two. Figure 2.1 illustrates alignment of sequence A to sequence B, where the vertical bars denote positions of matches and the hyphens represent insertions or deletions, commonly known as indels. Depending on data size, alignment may take several days of computation. For instance, short read alignment tools like MAQ [92] and SOAP [96] typically require more than 5 CPU-months and 3 CPU-years, respectively, for aligning 140 billion basepairs of data [86].

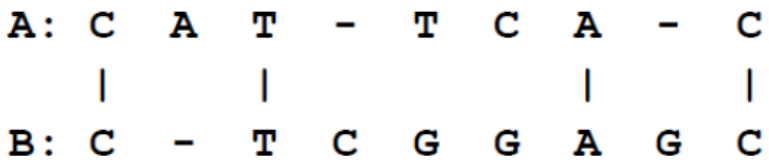


Figure 2.1: Genome alignment between sequences A and B showing matches (vertical bars), mismatches, and indels (hyphens).

Burrows Wheeler Aligner (BWA), one of the widely used alignment tools, implements Burrows Wheeler Transform (BWT) algorithm [21] to align short queries to reference genomes. It is lightweight, supports paired-end mapping, gapped alignment, and can work with various formats such as ILLUMINA [37] and ABI SOLiD [58]. By default, it generates alignment information in Sequence Alignment Map (SAM) for-

mat, which is convenient for further analysis such as visualization of alignment and variant detection [93]. Although it supports multithreading, the speedup attained from it is limited while processing large data sets.

2.2.2 Overview Of Variant Detection And HaplotypeCaller

Variants or mutations are nucleotides in query sequences that differ from the reference genome by a single base pair or a comparatively longer interval. In Figure 2.1, at the fifth position of the aligned sequences there occurs a mismatch or single nucleotide polymorphism (SNP). SNPs can account for the variation in traits among different individuals. They can also serve as markers in genome wide association studies (GWAS) and help in identifying genes associated with traits such as disease susceptibility. The performance of a variant calling tool is largely dependent on the quality and coverage of the data.

For our experiments, we use the state-of-the-art variant detection tool GATK's HaplotypeCaller. HaplotypeCaller creates an index for the input data and then locates variations between the query and reference. Once a difference is detected, it performs local assemblies to fill gaps or correct mistakes. In spite of its accuracy, the high computation time makes it prohibitive for large data sets.

2.2.3 Makeflow And Work Queue

Makeflow [10] is a workflow engine that defines a workflow in a directed acyclic graph (DAG) format. Being similar to Make, for each task, it requires a list of its input files, output files, dependencies, and the underlying command. Since both implicit and explicit requirements of the tasks are defined in the rules, the tasks can be sent to any platform that supports the executable. It is also fault tolerant as it can resume running the tasks after a crash.

Work Queue [19] is a master-worker job execution engine that can harness re-

sources from clusters, clouds, and grids. Once tasks are defined using Makeflow, execution is started by creating and broadcasting a Work Queue master. The master waits for workers and distributes tasks among them. Workers are created either at a local machine, or by submitting multiple workers at once to a batch system like Condor or SGE. Once the workers connect to the master, tasks are sent to the workers. The associated data are transported and cached at the workers. This caching strategy requires the data to be sent only once to each worker. It must be noted that although in a single workflow caching is beneficial, at the termination of the workflow the caches must be cleared for the purpose of security and efficient use. Hence, to achieve maximum benefit from caching, interrelated workflows can be merged to reduce the overhead of frequent data transfer across the network.

2.2.4 Data Partitioning

In distributed computing, a large workload is decomposed into multiple smaller, independent tasks that can be executed in parallel. Data partitioning refers to the method of splitting data in the given workload for parallel execution. We conceptually divide the steps of genome alignment and variant detection, and use the Makeflow language to define the taks and their dependencies. Makeflow can utilize resources from various systems, including multicore machines, batch systems like Sun Grid Engine (SGE) [51], and Condor Pool [100] to execute the tasks. For our experiments, we set up Makeflow to use the Work Queue master-worker framework as an alternative scalable infrastructure that can process data across clusters, clouds, and grids. Work Queue also handles all data transfers and can cache files when running jobs remotely.

We identify that a key feature for assuring improved runtime during parallelization is efficient partitioning of data. We explore different approaches of data partitioning for BWA and HaplotypeCaller and compare each combination using Work Queue-derived resources.

- For *granularity-based partitioning* in BWA, we test the workflow by varying the granurality, that is, number of partitions, and the size of each partition. We determine the optimal granularity value or partition size to be the one which incurs lowest runtime. For our query data comprising 140 million reads, splitting it into smaller chunks such that each contained 200000 reads proved to be the optimal choice. This resulted in 715 smaller files, which were then used for running BWA.
- For *individual-based partitioning* in BWA, we use coarser granularity by dividing the pooled data into multiple files, each corresponding to a sample or individual. As our data consists of genomic sequences from 50 individuals, we divide it into 50 files based on individual names.

Figure 2.2 illustrates the framework for implementing these data partitioning apparoaches in BWA. The split function refers to the creation of smaller query files based on a granularity value (for granularity-based partitioning) or individual names (for individual-based partitioning). In both cases, we assign the task of running BWA on each pair of small query data and reference data to a Work Queue worker. At the completion of alignment, we add read group and platform information to each BWA output (SAM file) and compress them into their sorted and indexed binary versions (BAM) for compatibility with the next step, that is, GATK's HaplotypeCaller.

In the next phase of the pipeline, we test three methods of partitioning aligned data for variant detection using HaplotypeCaller.

- For granularity-based partitioning, similar to BWA, we split the aligned data on the basis of an optimal granularity value.
- For individual-based partitioning, we split the aligned data based on individual names, resulting in 50 smaller files.
- For the new approach, called *alignment-based partitioning*, we partition the file (BAM) containing aligned output. We first split the reference genome into multiple smaller files, each containing a fixed number of unique contigs. For each small reference file, we split the pooled BAM file such that each small BAM file would contain alignment information of reads corresponding to the contigs of that particular reference subset.

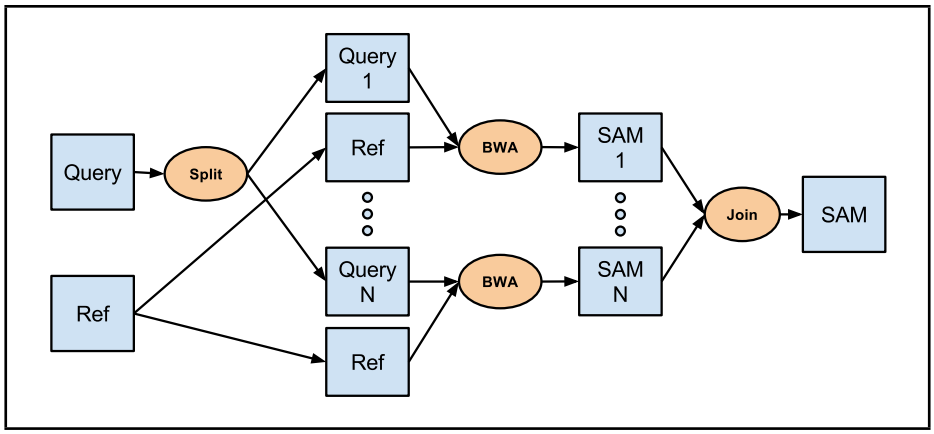


Figure 2.2: Framework of granularity-based and individual-based data partitioning approaches in BWA. For the test data set, N=715 for granularity-based and N=50 for individual-based. At the end, output files (SAM format) of BWA are joined to form a single file containing all the alignment information.

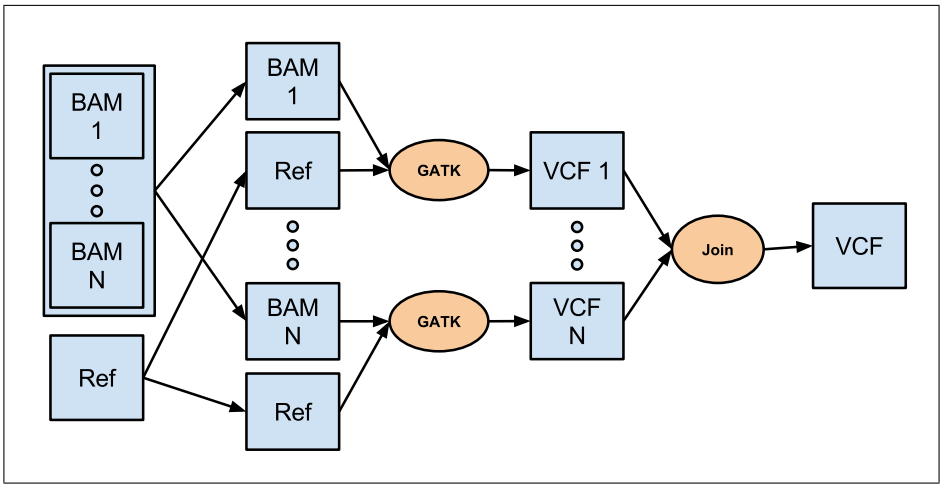


Figure 2.3: Framework of granularity-based and individual-based data partitioning approaches in HaplotypeCaller. For the test data, N=715 for granularity-based and N=50 for individual-based. The reference file and each sorted and indexed BAM file are sent to a worker for executing GATK's HaplotypeCaller. Outputs of HaplotypeCaller, in VCF format, are joined to create a single output file.

Figure 2.3 depicts the workflow for executing HaplotypeCaller using granularity-based and individual-based approaches. Figure 2.4 shows alignment-based data partitioning implemented in HaplotypeCaller. For each approach, we execute HaplotypeCaller on a pair of smaller sorted BAM file and the reference sequence on a Work

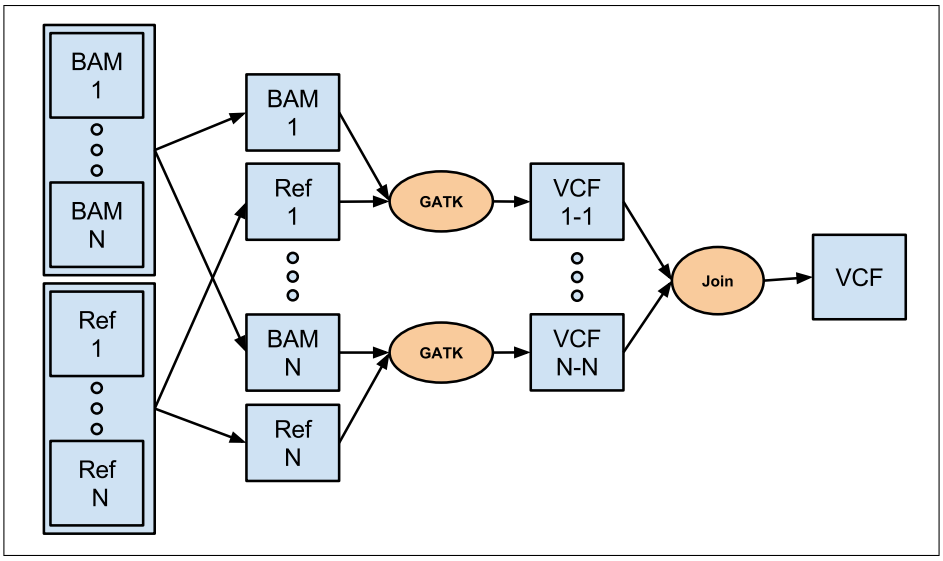


Figure 2.4: Framework of alignment-based data partitioning approach in HaplotypeCaller. The reference file was split into bins and the SAM file was split based on the contigs in the bins to which the reads aligned. Each pair of smaller reference bin and its corresponding BAM file were then sent to a worker to run GATK's HaplotypeCaller.

Queue worker. The output of HaplotypeCaller contains variant information in VCF format [38]. In the final step of the pipeline, we concatenate individual VCF files into a single VCF file containing variants for the entire population.

2.2.5 Workflow Fusion

We define workflow fusion as the idea of using information about the software and the computing systems, which are involved in the execution of a workflow, to accelerate a set of sequential but interrelated workflows. These cross workflow optimizations fuse the previously independent workflows into one. Though this causes less flexibility in how the workflows are used, the improvements can be worthwhile. As we look at ways in which workflow fusion can benefit performance, it can be noted that different methods require different levels of knowledge of the workflows and difficulty in the depth of fusion. We explore *full workflow caching* (requires concatenation of multiple Makeflow files), *choke elimination* (requires efficient partitioning of multiple

workflows), and *full fusion* (requires merging of two divergent partitioning techniques for acceleration).

2.2.5.1 Full Workflow Caching

A basic approach of workflow fusion is full workflow caching. This method relies on a single manager of tasks and data that allows for better data management. Although Work Queue aggressively caches all files that are sent to a worker, caching does not occur between independent workflows. By merging the workflows, we take full advantage of previously cached data that are useful for multiple steps. Figure 2.5 shows the organization of the cache controller, and how merging them unifies the controller. This is the most basic form of workflow fusion, as all that is needed is to merge two Makeflow files together and check that the reference, input, and output files are consistent to maintain the logical flow of the newly created workflow.

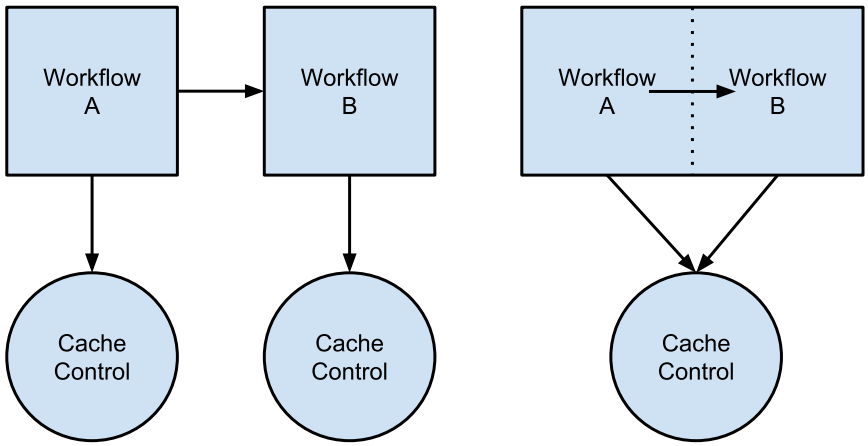


Figure 2.5: Shared cache controller among two merged workflows, A and B. In practice this is done using Work Queue, where all files in a Makeflow are aggressively cached. When separate, the two workflows' caches are independent and can not be used between each other. However, when the control of that cache is shared, previously transferred files can be utilized.

In this form of workflow fusion, we rely on the workflow execution engine for performance improvement. This method is the easiest to implement, and its benefits can be seen at all levels of workflow fusion, as illustrated in Figure 2.6. This benefit, however, is negated if files are modified between workflows. For example, the base reference is split during alignment-based partitioning for GATK, and is no longer recognized by Work Queue as a cached file. Although initially alignment-based GATK benefitted from limiting the amount of transferred data [32], this approach requires sending each subset of the reference, thereby losing the advantage of caching. Thus, combining two workflows presents challenges when the previously used partitioning scheme is no longer the optimal solution of the fused workflow.

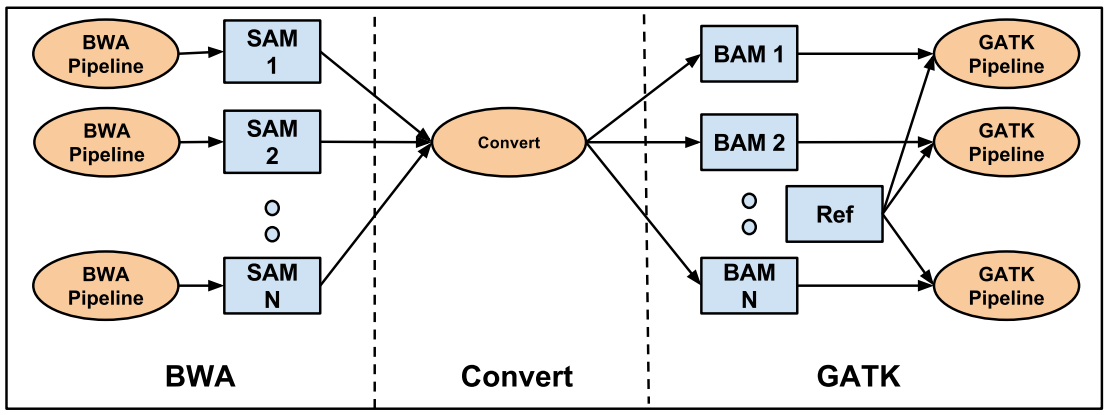


Figure 2.6: Framework of the Full Workflow Caching Concept. The pipeline comprises the BWA step, intermediate conversion steps for adding read groups, converting SAM files to their sorted and indexed formats, and the GATK step. The reference used by GATK is the same as that used in BWA, allowing for it to be cached at the worker and not sent as additional traffic.

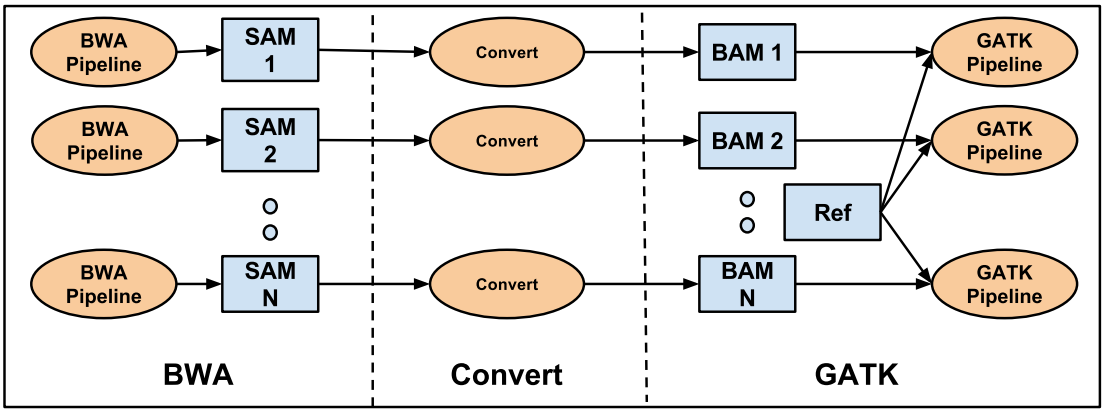


Figure 2.7: Framework of the Choke Elimination Concept. As can be seen here, the removal of intermediate data choke points allows computational threads to progress further through the workflow without needing to wait for slower threads to finish and communicate with the master. This lowers the amount of data the master is required to deliver at once and allows the transfers to be offset based on when they arrive.

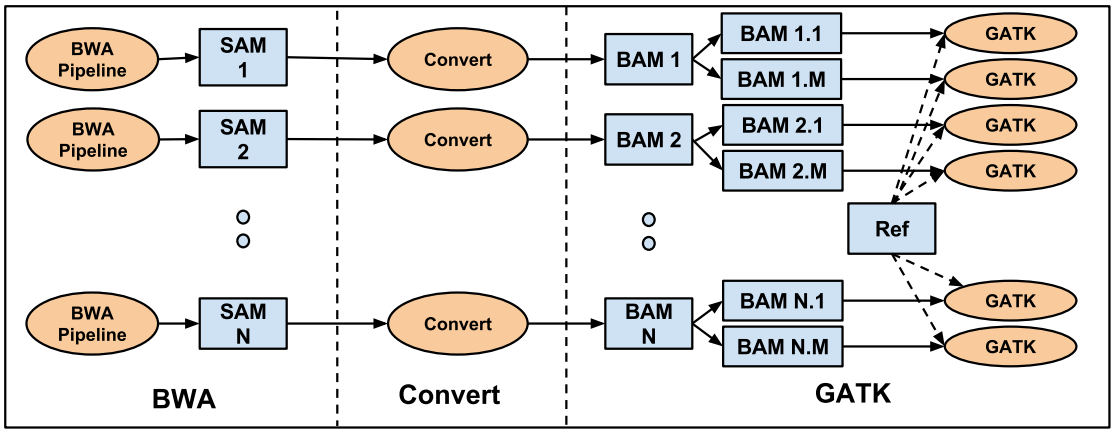


Figure 2.8: Framework of the Merged Partition Concept. As can be seen here, the conversion step appears prior to the mapping, but could go either before or after depending on what state the inputs need to be. It is also useful to note how, though as simple map is used here, any process that relates two partitioning methods could be added as long as it does not require coalescing the full data set.

2.2.5.2 Elimination Of Choke Points

A choke point is simply a location in the workflow that requires all threads of computation to converge. Similar to a barrier operation in traditional parallel computing,

choke points hinder computation by stopping all concurrency until the slowest serial task completes. The effort of eliminating these choke points is the second approach of workflow fusion. When eliminating choke points, an effort is made to alleviate traffic where data converges on a single node and is redistributed in some form. Although caching can limit some of the outgoing traffic from later partitioning steps, there is still considerable traffic that is incurred while sending data in and out of these choke points.

As choke point elimination strives to take advantage of files already distributed, it is important to understand how to convert the results of the previous workflow into inputs of the next workflow. Such transformations are crucial, since they can be applied directly to the data. A clear example of a transformation is the sorting of aligned results prior to starting variant calling. Prior to fusion, all of these processes are sequentially completed, but performing a transform as part of the prior step allows avoiding expensive traffic to converge the data on one system.

This is done most clearly by utilizing the partitions of the previous workflow as the partitions of the subsequent workflow, as shown in Figure 2.7. Using established partitioning eliminates intermediate partition transfers, and allows independent partitions to continue executing without waiting for slower branches. The cost of this approach is that both workflows must be able to utilize this partition with acceptable performance. If the two different workflows do not share a common suitable partitioning scheme, we can utilize the information about the workflows to potentially find a more appropriate hybrid method.

2.2.5.3 Merged Partitioning

Finding a hybrid partitioning scheme requires knowledge about how different workflows interact and therefore how they could be optimally partitioned. Unfortunately, it is not always obvious how to do such divisions. For example, alignment-

based partitioning is best for GATK but this requires performing alignments using BWA. One compromise is to maintain BWA partitions utilizing individual-based partitioning in BWA, and then partioning each individual utilizing alignment-based method. Although this is a basic method to merge partitioning schemes, the idea can be extended to any method that relates one set of inputs to another, as shown in Figure 2.8.

When successful, applying these ideas to a set of workflows alleviates the time and stress on the system, while allowing faster machines to process more without having to wait and proceed in lock step with remaining tasks.

2.2.5.4 Potential Issues Of Workflow Fusion

Although performance is often improved by merging multiple workflows together, there are several pitfalls. The first is that as the workflows become more intertwined, debugging becomes more difficult to monitor, understand, and fix. Undetected failures in different states produce incorrect output, which could have occurred at any of the fused processes. This issue can be exacerbated when workflows operate at different stages concurrently. For this reason a clear understanding of what errors may occur within the different processes and how to handle them is paramount. The second pitfall is some methods of partitioning, or combinations of partitioning methods, may not accelerate the workflow or be feasible due to dependencies (e.g., alignment-based GATK drawn from BWA output). When this occurs, more time must be spent in understanding and writing appropriate transformations.

2.3 Results And Discussion

For our experiments, we align and detect variants in 50 samples of *Quercus rubra* (northern red oak), sequenced by ILLUMINA HiSeq RAD data [111]. The computational set-up consists of a cluster of 26 machines, each with 8 cores and 32 GB

RAM. To illustrate the base runtime, BWA requires about 4 hours to complete the alignment procedure sequentially, and GATK's HaplotypeCaller requires 12 days to detect SNPs and indels. Runtime for BWA does not include the time for indexing the reference, which can be done once and reused for each alignment task.

2.3.1 Optimal Data Partitioning

First, we explore the methods of data partitioning for BWA: granularity-based and individual-based. For the test data set, granularity-based partitioning performs well, by only requiring 13 minutes to parse the entire data into 715 partitions. This is expected as the input data is simply decomposed into contiguous, non-overlapping subsets based on the optimal granularity size. On the other hand, individual-based method incurs an additional time to look up the set of barcode information for identifying the individual or sample names, based on which partitioning is done. It requires 18 minutes to create 50 partitions. A comparison of runtimes can be seen in Table 2.1.

Next, we test different partitioning methods proposed for GATK. Granularity-based approach partitions the aligned output of BWA in less than an hour. Individual-based requires 27 minutes to create 50 partitions corresponding to 50 individuals. Alignment-based partitioning splits the reference and alignment file into 10 partitions based on the underlying alignment information. This method requires 15 minutes. A comparison of results can be seen in Table 2.2.

2.3.2 Tool Improvement

Table 2.1 presents the runtimes for BWA using different data partitioning approaches. Alignment and concatenation stages of granularity-based and individual-based approaches require similar time. The step of splitting the query in individual-

TABLE 2.1

COMPARISON OF RUNTIMES FOR DIFFERENT APPROACHES OF DATA PARTITIONING IN BWA

BWA Runtime (DD:HH:MM:SS)				
Partitioning	**Split**	**Align**	**Concatenation**	**Total**
Individual	18:39	17:58	22:26	59:23
Granular	13:23	21:11	21:10	**55:44**

TABLE 2.2

COMPARISON OF RUNTIMES FOR DIFFERENT APPROACHES OF DATA PARTITIONING IN GATK'S HAPLOTYPECALLER

GATK Runtime (DDd HH:MM:SS)					
Partitioning	**Coupling**		**Variant Calling**	**Concat**	**Total**
	Split	**Other**			
Individual	27:35	3:59:10	2d 5:18:23	9:00	2d 9:54:08
Granular	52:34	7:06:25	41:51	5:30	8:46:20
Alignment	15:41	1:03:47	24:36	4:00	1:48:04

based partitioning is slower due to higher look up time in searching the barcode information. As seen in Table 2.2, for HaplotypeCaller, preparation of input involves splitting the aligned data (SAM format) from BWA, adding intermediate information (read groups) and converting each subset to its sorted and indexed format. The corresponding runtime for alignment-based partitioning is significantly lower than the other approaches.

TABLE 2.3

GRANULARITY-BASED BWA REQUIRES LESS TIME WHEN MORE WORKERS ARE IN USE

Runtime (HH:MM:SS)						
Workers	**Data 1**			**Data 2**		
	Time	**Speedup**	**Efficiency**	**Time**	**Speedup**	**Efficiency**
2	3:01:57	1.94	0.97	8:46:13	1.80	0.90
5	1:14:30	4.77	0.95	3:33:45	4.45	0.89
10	1:08:42	5.19	0.51	1:48:10	8.78	0.87
20	59:00	5.98	0.29	55:34	17.23	0.86
50	57:06	6.19	0.12	33:17	28.27	0.57
100	56:00	6.30	0.06	20:52	47.41	0.47

Based on these results, we infer granularity-based data partitioning to be the best strategy for BWA. We test its runtime behavior, particularly speedup and efficiency, by scaling up the number of workers used in the experimental set-up. Table 2.3 presents the framework's performance in response to workers, ranging between 2 and 100. Using more workers does not guarantee a proportionate improvement in speedup. For our test data and machine configuration, it does not scale linearly beyond 20 workers. The primary reason is the overhead incurred in data transfer, especially sending the reference, query, and index files to each worker. In Figure 2.9, we show the histogram of runtime for executing 715 tasks in each stage (bwa aln and samse) of BWA. It explains the overall behavior of granularity-based BWA tasks. Figure 2.10 provides a more detailed visualization of the runtime behavior of BWA when using 100 workers. The number of tasks submitted and completed follow a similar pattern until the former reaches a plateau denoting submission of all tasks.

Figure 2.11 demonstrates data transfer from the master to workers over the network. This is a desirable feature in an environment without a shared file system, as in our case. The transferred data can be cached at the workers and reused for later stages of the pipeline.

As seen in Table 2.2, for HaplotypeCaller, alignment-based approach is most efficient. Here, while splitting the aligned data, we optimize the method of searching reference sequence for variants. Instead of exploring the entire reference file, we restrict the algorithm to a specific subset of the file that is guaranteed to bear pertinant information. In this process, we also get rid of nuisance entry for unaligned data, thereby further filtering the search space. Finally, we enable reduction in data transfer as now only a subset of the reference needs to be processed at each worker, not the entire set of reference sequences.

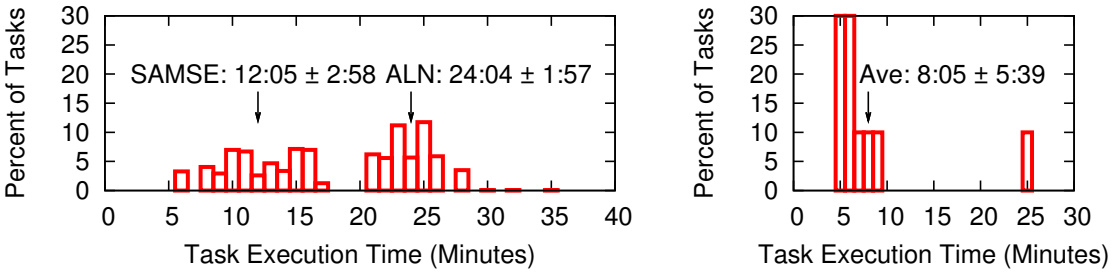


Figure 2.9: Left: Histogram of runtime for BWA is bimodal as it comprises two steps: ALN (alignment) and SAMSE (generation of aligned output in SAM format). Times are measured for 1430 tasks, with 715 attributing to each step. Right: Histogram of runtime for GATK exhibits tight coupling, except for a single heavy outlier.

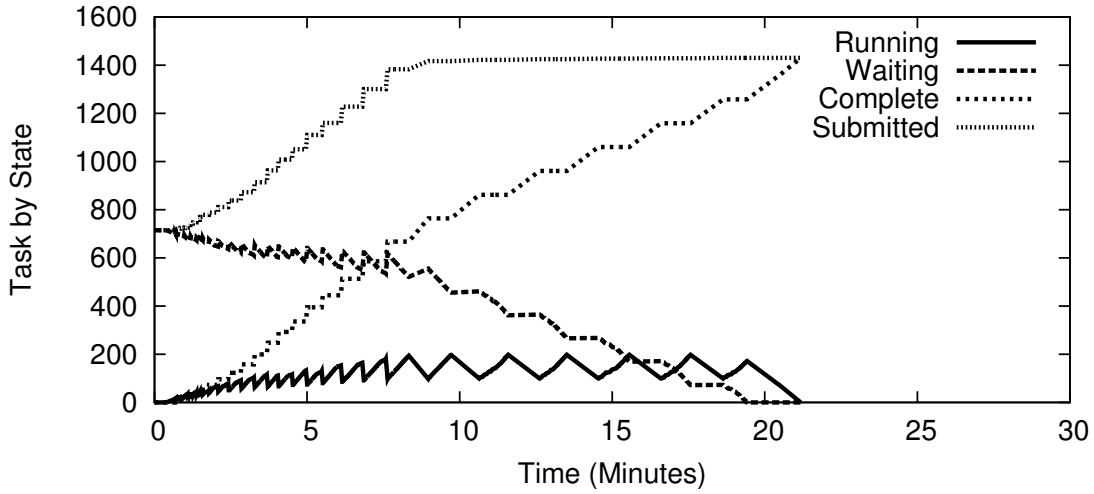


Figure 2.10: Runtime behavior of granularity-based BWA using 100 workers. The continuous line represents currently running tasks, the jagged appearance is due to the manner in which tasks are distributed using Work Queue. While distributing tasks, finished tasks wait to be collected and are then sent out. This waiting causes the jaggedness.

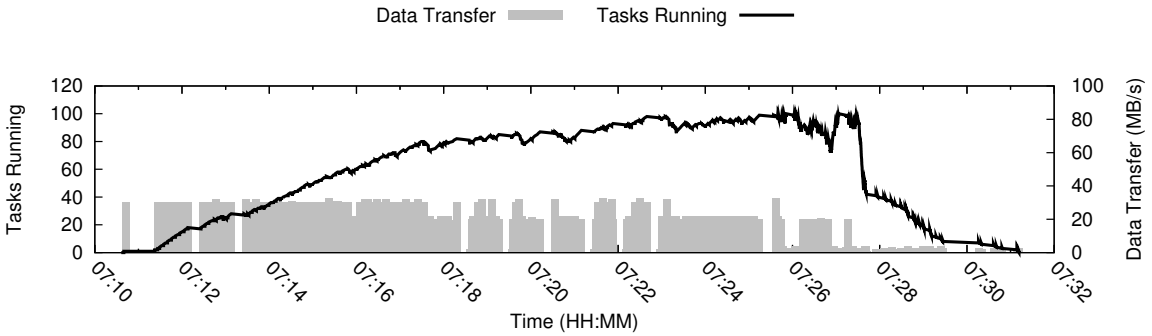


Figure 2.11: Mechanism of data transfer for an environment without a shared file system. The thick line denotes the number of tasks executing during the given timeline for granularity-based BWA. The gray bars show data transfer from master to worker nodes. The left axis measures number of tasks running whereas the right axis measures the rate of data transfer in MB/s.

2.3.3 Pipeline Improvement

Based on the above results, we combine granularity-based BWA and alignment-based HaplotypeCaller to generate an optimized data analysis pipeline for genome

data, as shown in Fig. 2.12. Table 2.4 provides the runtimes specific to our pipeline (using 100 Work Queue workers) in a heterogeneous, distributed system. It shows the breakdown of runtimes for individual components of the pipeline. Efficient data partitioning followed by parallelization of tasks reduces runtime of each phase, resulting in an overall improvement from 12 days to under 3 hours.

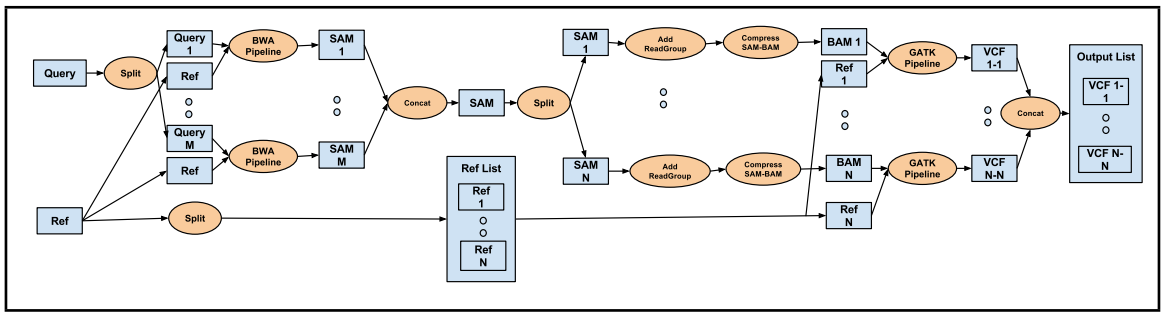


Figure 2.12: Framework of the optimized pipeline incorporating granularity-based BWA and alignment-based HaplotypeCaller. It also includes the intermediate stages of adding read groups to SAM files and converting them to their sorted, indexed, and binary formats.

2.3.4 Optimal Workflow Fusion

The first method that we perform is a full cache fused workflow. This is done by concatenating the Makeflow files for each workflow. Following this, all intermediate conversion processes are added, and the fused makeflow is run utilizing Work Queue. Table 2.5 compares the times for running the pipeline in a sequential order against the parallel implementation of our proposed framework.

The second approach is choke point elimination, that is implemented with a common partitioning scheme. We utilize Makeflows written for individual-based parti-

TABLE 2.4

RUNTIMES OF THE STEPS IN THE OPTIMIZED PIPELINE

Runtime (MM:SS)		
BWA	**Split Query**	13:23
	Parallelized BWA	21:11
	Concat SAM	21:10
Coupling	**Split Ref and SAM**	15:41
	Add RGs	11:10
	SAM to BAM	52:37
GATK	**Parallelized GATK**	24:36
	Concat VCF	4:00
Total		2:43:48

TABLE 2.5

COMPARISON OF RUNTIMES FOR SEQUENTIAL, PARALLEL, AND DIFFERENT SCHEMES OF WORKFLOW FUSION EXECUTION OF THE PIPELINE

Runtime (HH:MM:SS)				
	BWA	**Coupling**	**GATK**	**Total**
Sequential	4:04:00	6:50:41	12d 00:00:00	12d10:54:41
Parallel	55:44	1:23:28	24:36	2:43:48
CF	2:42:49			2:42:49
CF + CE	2d 8:43:46			2d 8:43:46
CF + CE + PF	1:55:37			1:55:37

tioning in BWA and GATK. This method requires almost 2 and a half days,(Table 2.5), which is on the same scale as BWA and GATK's separate makeflow using individual-based partitioning. This implementation improves coupling of workflows by eliminating the join-split sequence used in the pipeline and cache fused approach. However, the results are much slower, due to GATK's sluggish performance on this partition method.

The final method is full partition merging of BWA and GATK. We begin with the Makeflow of individual-based BWA and remove the concatenation step. Each individual is then split using alignment-based method. Each new split has the intermediate processes performed. It adds a significant amount of tasks, with only a small amount of additional overhead, as the inputs are smaller. This method reduces the overall runtime to less than 2 hours (Table 2.5). This is an improvement over the best results obtained from optimal data partitioning alone. Hence, optimal caching and higher level of concurrency cause a significant improvement in performance (Figure 2.13). Figure 2.14 further illustrates the significant difference between coupling steps of cache fused and partition fused approaches of workflow fusion.

2.3.5 Application Of The Data Analysis Framework

Our data analysis framework is being used to study the genomes of multiple organisms, including the economically and ecologically valuable plant species *Quercus rubra* (northern red oak). It is an important plant in forest communities, spanning across a wide range of ecosystems. Oak forests are threatened by diseases, pests, and poor management practices. Development of a dense genetic map of *Q. rubra* will help in understanding the impact of these challenges and devising potential measures to combat them. A genetic map is an inference of the linear order of specific DNA sequences based on the observed recombination events in the progeny of known parents or grandparents. They are used to relate genotypes to phenotypes

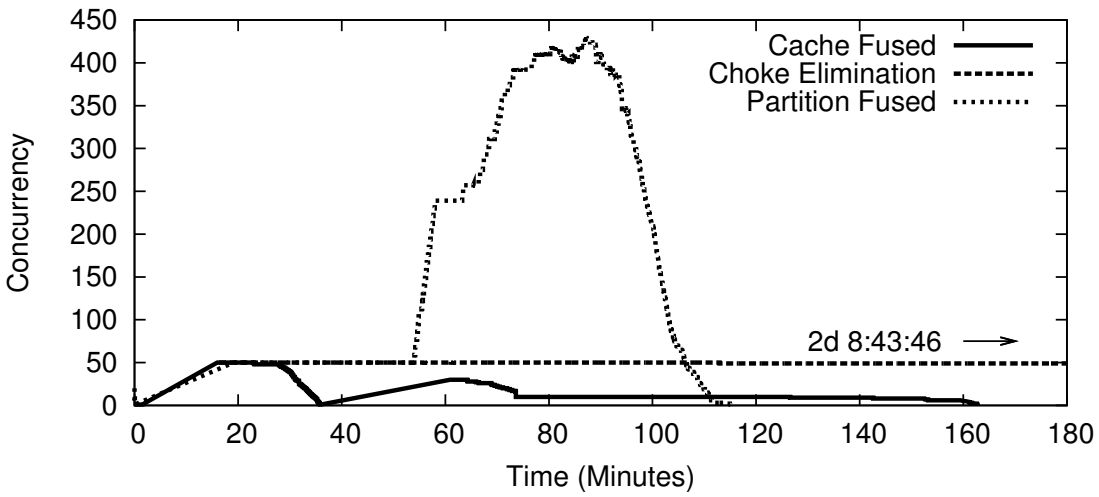


Figure 2.13: Comparison of available concurrency for Partition Fused, Choke Elimination, and Cache Fused workflows. Available concurrency refers to the tasks that are either ready to be run or running. Partition Fused method increases the number of partitions, thereby allowing a higher level of concurrency.

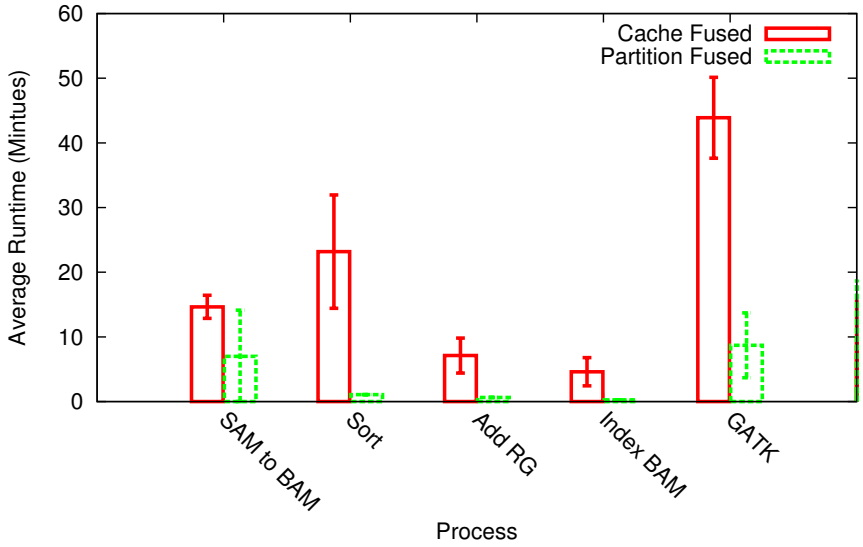


Figure 2.14: Comparison of average runtimes for different steps of coupling in cache fused and partition fused workflows. The time spent in intermediate steps and GATK are higher in cache fused than partition fused. It is important to note that the number of partitions was 10 for cache fused and 500 for partition fused.

and to map the contigs from whole genome sequencing data to linkage groups. We use our proposed framework to identify variants or SNP-based markers necessary to create a high-quality, dense genetic map of *Q. rubra*, the first map for the section of oaks, the Lobatae. (Figure 2.15). This map will be useful in recognizing the genetic structure responsible for its adaptive evolution and tolerance to stress.

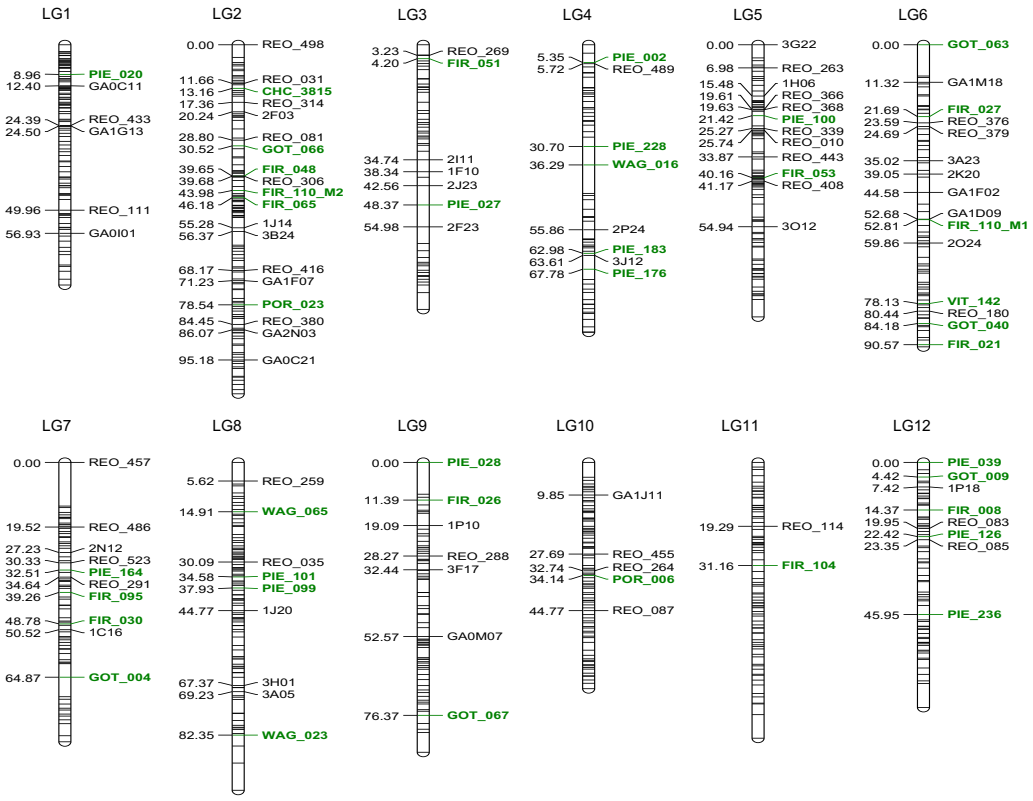


Figure 2.15: Genetic map of nothern red oak (*Quercus rubra*) corresponding to 12 chromosomes or linkage groups (LGs). Our proposed data analysis framework was used to generate SNP-based markers for constructing the map.

The genetic map of *Q. rubra* and the methods used in its construction have been published in the following peer-reviewed journals:

- A. Konar, **O. Choudhury**, T. McCleary, S. Schlarbaum, O. Gailing, M. Coggeshall, S. Emrich, M. Staton, M. Pfrender, J. Carlson, J. Romero-Severson. *High-quality genetic mapping with ddRADseq in the non-model tree Quercus*

rubra. Accepted, BMC Genomics, 2017.

- A. Konar, **O. Choudhury**, O. Gailing, M. Coggeshall, M. Staton, S. Emrich, J. Carlson, J. Romero-Severson. *A Genetic Map for the Lobatae*. lnternational Oaks, No. 27, 2016.

2.4 Conclusion

Analysis of massive genome data requires days or weeks of computation. The extent of performance improvement that can be achieved from sophisticated algorithms, data structures, or multithreading is often limited. Designing a framework that can efficiently process large-scale data by harnessing resources from clusters, clouds, and grids is a more tractable solution for this problem.

Here, we design an optimal pipeline or data analysis framework for the widely used genomics applications: genome alignment and variant detection. To construct this framework, we propose and explore various strategies of data partitioning and workflow fusion. Through a comparative study, we determine optimal methods of partitioning and workflow fusion, which significantly improve performance of the framework when tested on real data sets. For the test data and applications, the optimal framework reduce the processing time from 12 days to under 2 hours. The efficacy of this framework encouraged us to apply it in several genomic projects, particularly in studying genomic data of complex organisms like malarial mosquitoes and plant species.

CHAPTER 3

COMPUTATIONAL RESOURCE OPTIMIZATION FOR EXECUTING DATA-INTENSIVE GENOMIC APPLICATIONS ON CLUSTERS AND CLOUDS

The following manuscript describes the work in this chapter.:

- O. Choudhury, D. Rajan, N. Hazekamp, S. Gesing, D. Thain, S. Emrich. *Balancing Thread-level and Task-level Parallelism for Data-Intensive Workloads on Clusters and Clouds*. IEEE International Conference on Cluster Computing, 2015.

3.1 Background

Determining an optimal runtime configuration of parallel and distributed applications is a challenging task. To tune an application on a particular system, the end-user is confronted with a wide array of controls: number of machines, number of cores per task, partitioning the data, task scheduling strategies, and so forth. It is customary in academic papers to perform an exhaustive exploration of these parameters, and then select a configuration which is optimal under some limited set of circumstances. Any change to the machines, network, or workload itself could result in poor performance, often orders of magnitude worse than what is achievable. From the user's perspective, running an application multiple times to obtain a performance curve is a tedious process.

Therefore, when designing distributed applications, our objective should be to achieve acceptable performance the first time by avoiding extremely bad configurations. It is impossible to provide a comprehensive model for programs of arbitrary structure (this is simply the halting problem). Fortunately, users in a given area of

study often use similar computational patterns over and over again, which gives us an opportunity to develop a model that is highly effective within a given application domain. For example, in the realm of bioinformatics applications, many tools have the form of a database search: a program performs a search for a query pattern within a database of genomic strings. These tools are typically transformed into parallel applications by partitioning the query and/or the database, and then running multiple instances of the local multithreaded program in parallel.

We present a regression model-based technique to infer optimal runtime configurations and predict execution time and resource consumption for data-intensive workloads. We perform black-box analysis of the multithreaded program (computational kernel) to determine its runtime and memory usage relative to sizes of query, database, and local concurrency. Once obtained, this model is embedded in a more generic model of the distributed system that incorporates data partitioning, data movement, and the trade-off between local and distributed concurrency.

We show the effectiveness of our proposed approach in three commonly used bioinformatics applications BWA [89], Bowtie2 [81], and BLASR [26] that are widely used in production to study cancer [143] and variation-detection in general [98]. In all the cases, our proposed models accurately predict execution time and memory footprint. The optimal configurations identified by the models also enable cost-efficient utilization of commercial cloud-based resources like Amazon EC2 [1] and Microsoft Azure [2].

3.2 Methods

We develop models to estimate the behavior, particularly runtime and memory usage, of applications when executed in a distributed system. For our experiments, we use the short read alignment tools BWA, Bowtie2 and the long read aligner BLASR as generalized test applications. Millions of next generation sequencing (NGS) [127]

data, called reads, are generated by high-throughput sequencing techniques. The reads are aligned or compared to the genome of a related species, known as the reference genome, for further biological analysis. BWA and Bowtie2 implement Burrows Wheeler Transform (BWT) [22] method of data compression to align short reads, usually hundred bases long, to a reference sequence. We use the genome of mosquito *Culex quinquefasciatus* [107] as reference and simulate sequences as query data for BWA and Bowtie2. The reference genome contains 3171 contigs and is 562 MB in size. The primary advantage of simulating sequences is the ability to generate arbitrarily large queries for testing the model. We chose the query data to be 10x the reference based on a previous large-scale mosquito variant discovery project [116]. BLASR, on the other hand, aligns Single Molecule Sequencing (SMS) reads of size thousands of bases with high insertion and deletion errors to a reference genome. We use the *Anopheles gambiae* S form genome [84] containing 13042 scaffolds and size 230 MB as reference and PacBio sequenced real data set of size 1 GB as query for BLASR.

We develop a distributed version of the test applications using Work Queue [20], a master-worker framework that is flexible in varying different parameters, such as number of tasks, memory usage, cores used by each task when executed across a distributed system. We use a 12-core machine with 64 GB memory, x86 architecture as the master server and computing resources through the Sun Grid Engine (SGE) batch system as workers. The computing nodes in the grid engine were based on a shared computing platform and allowed running other applications. Our proposed model comprises two tiers - an application-level to describe the performance of user-level applications, and a generic system-level to predict the behavior of a distributed system when executing the applications. As the first model is application-specific, for all the test tools, it requires prior information about the sizes of reference and query data, and number of threads used to execute in multithreaded mode. The

second stage of the model requires granularity value that is defined as the number of tasks the workload is divided into when executed across a distributed system. It also depends on the underlying available resources, namely number of CPU cores, memory size, disk size, and network bandwidth.

For the application-level model, we record runtimes and memory consumed by BWA, Bowtie2, and BLASR for different configurations of input size and threads. We collect 343 data points by varying reference size, query size, and number of threads. Similarly, for the system-level model we vary number of tasks and number of cores used by each task to obtain data for runtime and memory usage. We randomly select $\frac{2}{3}^{\text{rd}}$ of the data for training the models and the rest for testing. We design regression models for runtime and memory usage based on the parameters that define the applications' behavior. Equations (3.1) and (3.4) are the application-level models for runtime and memory, respectively. Equations (3.13) and (3.17) define the system-level models for time and memory, respectively.

From the training set of 230 data points, we randomly select unique subsets of size 100 each. For each subset, we train the regression models to compute the values of regression coefficients $\beta^*, \gamma^*, \eta^*$ and ϕ^* in equations (3.3), (3.6), (3.16), and (3.19), respectively. We use these regression coefficients in the test data and compare model-predicted values with empirical values to measure its prediction accuracy.

3.2.1 Design Of Application-Level Model

3.2.1.1 Model For Runtime

Let R, Q be the sizes of the reference and query sequences (in GB), respectively and N be defined as the number of cores used for its execution. The runtime of a BWA, Bowtie2 or BLASR task, denoted as $T(R, Q, N)$, can be described by the following equation,

$$T(R, Q, N) = \beta_1 \frac{RQ}{N} + \beta_2 \tag{3.1}$$

where $\beta = \begin{bmatrix} \beta_1 & \beta_2 \end{bmatrix}^\top$ are regression coefficients for the model. The scalar β_2 signifies the fraction of the program that cannot be parallelized, as discussed in detail in section 3.3.1.

We briefly describe the method for obtaining optimal β as follows. Suppose the i^{th} measurement of runtime is denoted by $T(R_i, Q_i, N_i)$. Then we define,

$$\mathcal{T} = \begin{bmatrix} T(R_1, Q_1, N_1) \\ \vdots \\ T(R_n, Q_n, N_n) \end{bmatrix},$$

where $\mathcal{T} \in \mathbb{R}^n$ contains n measurements. Corresponding to each measurement, suppose we select basis functions of the form $\{\frac{RQ}{N}, 1\}$. Then the i^{th} equation for runtime generated with the data R_i, Q_i, N_i is written as,

$$T(R_i, Q_i, N_i) = \beta_1 \frac{R_i Q_i}{N_i} + \beta_2$$

With n such measurements, we construct the following matrix,

$$\underline{\mathrm{H}}_1 = \begin{bmatrix} \frac{R_1 Q_1}{N_1} & 1 \\ \vdots & \vdots \\ \frac{R_n Q_n}{N_n} & 1 \end{bmatrix}$$

Note that $\underline{\mathrm{H}}_1 \in \mathbb{R}^{n \times 2}$ The following linear regression equation can now be posed,

$$\mathcal{T} = H_1 \beta \tag{3.2}$$

It is known [30] that the solution to this equation is given by,

$$\beta^* = H_1^\dagger \mathcal{T}, \tag{3.3}$$

where $H_1^\dagger = (H_1^\top H_1)^{-1} H_1^\top$ is the Moore-Penrose pseudo-inverse [9]. Note that β^* denotes the optimal regression coefficients for our model in equation (3.1).

3.2.1.2 Model For Memory Usage

As the memory-usage of BWT-based aligners is independent of the number of reads to be aligned or the size of the query data [90], we observe that the memory consumption of our test applications is not considerably impacted by the query input. Hence, using previous notations, the model to estimate the memory usage of BWA, Bowtie2, BLASR can be designed by the following equation,

$$M(R, N) = \gamma_1 R + \gamma_2 N \tag{3.4}$$

where $\gamma = \begin{bmatrix} \gamma_1 & \gamma_2 \end{bmatrix}^\top$ are regression coefficients.

Similar to the model for time, we can write the following:

$$\mathcal{M} = \begin{bmatrix} M(R_1, N_1) \\ \vdots \\ M(R_n, N_n) \end{bmatrix},$$

$$M(R_i, N_i) = \gamma_1 R_i + \gamma_2 N_i$$

$$\underline{\mathrm{H}}_2 = \begin{bmatrix} R_1 & N_1 \\ \vdots & \vdots \\ R_n & N_n \end{bmatrix}.$$

Here, $H_2 \in \mathbb{R}^{n \times 2}$. Similar to equations (3.2) and (3.3):

$$\mathcal{M} = H_2 \gamma \tag{3.5}$$

$$\gamma^* = H_2^{\dagger} \mathcal{M} \tag{3.6}$$

3.2.2 Design Of System-Level Model

Following the design of the application-level model, we develop a model to understand the behavior when the initial workload is executed across a distributed system. In this case, the workload is first split into a number of smaller, independent tasks, each of which is then sent to and executed on the nodes of a cloud, cluster, or grid environment. The model to estimate system-level behavior is a generic model in which we can plug in the characteristics, that is, number of tasks and number of nodes used by each task, of the application-level model. The model for runtime also depends on the underlying hardware specifications, like speed of the processor, speed and size of the memory, speed and size of the disk.

3.2.2.1 Model For Runtime

Suppose T_S is the time taken to locally split a workload into a given number of tasks. Let T_{In} be the time the master server spends in transferring all the dependencies and inputs to the workers set across a distributed system and T_{Out} be the time required to send all the outputs back to the master. If T_J is the time for joining all the output files at the master and T_{Tasks} defines the total time required to execute K tasks, then the total runtime, T_{total}, of any workload executed across a distributed system can be expressed as:

$$T_{total} = T_S + T_{In} + T_{Tasks} + T_{Out} + T_J \tag{3.7}$$

Here, T_{Tasks} depends on the number of tasks K the initial workload is split into and the number of cores N allocated for each task. Each task aligns a subset of query ($\frac{Q}{K}$) to the reference sequence R. Let us assume M is the number of available machines and C is the number of cores present in each machine. We can define T_{Tasks} as

$$T_{Tasks} = \eta_3 T\left(R, \frac{Q}{K}, N\right) \times \frac{KN}{MC}, \tag{3.8}$$

where KN represents the total number of cores required for K tasks if each needs N cores. MC indicates the total number of available cores. For a finite number of usable cores P, we can determine optimal K and N to minimize T_{Tasks}. We can write the i^{th} equation for T_{Tasks} as:

$$T_{Tasks_i} = \eta_3 T(R, \frac{Q}{K_i}, N_i) \times \frac{K_i N_i}{MC},$$

such that $K_i N_i = P$, and $N_i \leq C$. We can constrain P as $P \leq MC$, if the total number of required resources is within the range of available resources, in which case optimal parallelism can be attained. On the other hand, if $P > MC$, that is, the workload requires more resources than what is readily available, resources can be re-used, which often limits the extent of parallelism. Thus, a key feature in minimizing the overall time T_{total} in equation (3.7) is to determine optimal values of K and N.

The times required for decomposing the workload (T_S) and joining the individual output files (T_J) at the master are dependent on the size of data being used in each case and speed and size of the disk. We split the query into K smaller subsets and align each to the reference sequence [31]. Once individual tasks are completed successfully, the outputs are joined to generate results identical to the applications' sequential implementation. The time for splitting a query of size Q into K partitions is:

$$T_S = \eta_1 \frac{QK}{D} \tag{3.9}$$

where D is the disk read-write speed.

Let the size of output file returned to the master be denoted by O. Then we can express T_J as:

$$T_J = \eta_5 \frac{OK}{D} \tag{3.10}$$

T_{In} and T_{Out} vary with the amount of data being transferred over the network and the corresponding network bandwidth. For T_{In}, as the smaller query file (of size $\frac{Q}{K}$) is unique for each task, $\frac{Q}{K} \times K$ or Q amount of data must be sent to the workers. The shared reference file can be cached at the instances to reduce transfer time such that new tasks re-using an instance can utilize the data. Given an infinite number of machines, each containing C-cores, the number of machines required to execute K tasks, each using N cores, is $\frac{K}{\frac{C}{N}}$. Assuming the sizes of other dependency files to be transferred is negligible compared to the input data and the total memory required by all tasks using a given instance meets the upper bound of the memory available at that instance, we can express T_{In} as:

$$T_{In} = \eta_2 (\frac{Q}{B} + \frac{RKN}{BC}) \tag{3.11}$$

where B is the network bandwidth.

Similarly,

$$T_{Out} = \eta_4 \frac{O}{B} \tag{3.12}$$

Using equations (3.8)–(3.12), we can re-write equation (3.7) as (3.13). The component $T(R, \frac{Q}{K}, N)$ in (3.13) is defined previously in (3.1).

For a given workload (R, Q), hardware specifications (B, D), machine configura-

$$\tau = \eta_1 \frac{QK}{D} + \eta_2(\frac{Q}{B} + \frac{RKN}{BC}) + \eta_3 T(R, \frac{Q}{K}, N) \times \frac{KN}{MC} + \eta_4 \frac{O}{B} + \eta_5 \frac{OK}{D} \tag{3.13}$$

$$\tau_i = \eta_1 \frac{QK_i}{D} + \eta_2(\frac{Q}{B} + \frac{RK_iN_i}{BC}) + \eta_3 T(R, \frac{Q}{K_i}, N_i) \times \frac{K_iN_i}{MC} + \eta_4 \frac{O_i}{B} + \eta_5 \frac{O_iK_i}{D} \tag{3.14}$$

tion (M, C), and available resources $(P = K_iN_i)$, we detrmine optimized values of K and N such that the overall completion time is minimized.

Similar to the previous cases, we can write the following:

$$\mathcal{T}' = \begin{bmatrix} T_{total1} \\ \vdots \\ T_{totaln} \end{bmatrix},$$

$$\underline{\mathrm{H}}_3 = \begin{bmatrix} K_1 & K_1N_1 & T(R, \frac{Q}{K_1}, N_1)K_1N_1 & O_1 & O_1K_1 \\ \vdots & & \ddots & & \vdots \\ K_n & K_nN_n & T(R, \frac{Q}{K_n}, N_n)K_nN_n & O_n & O_nK_n \end{bmatrix}$$

Here $\underline{\mathrm{H}}_3 \in \mathbb{R}^{n \times 5}$. Thus,

$$\mathcal{T}' = H_3\eta, \tag{3.15}$$

where $\eta = \begin{bmatrix} \eta_1 & \dots & \eta_5 \end{bmatrix}^\top$ are the regression coefficients. The optimal coefficients are given by

$$\eta^* = H_3^\dagger \mathcal{T}'. \tag{3.16}$$

3.2.2.2 Model For Memory Usage

In order to harness resources from a distributed system, a workload, in this case the query sequence, is first split at the master machine. The memory requirement of this procedure is proportional to the query size. Once all the tasks are executed at the workers, the outputs are sent back to the master for joining. For this, the

memory usage depends on the amount of data being joined, alternatively on the sizes of query and reference sequences. Thus, the memory needed at the master server $M_{Master}(R, Q)$ can be defined as:

$$M_{Master}(R, Q) = \phi_1 R + \phi_2 Q \tag{3.17}$$

where $\phi = \begin{bmatrix} \phi_1 & \phi_2 \end{bmatrix}^\top$ are regression coefficients.

As shown above,

$$\mathcal{M}' = \begin{bmatrix} M_{Master}(R_1, Q_1) \\ \vdots \\ M_{Master}(R_n, Q_n) \end{bmatrix},$$

$$M_{Master}(R_i, Q_i) = \phi_1 R_i + \phi_2 Q_i$$

$$\underline{\mathrm{H}}_4 = \begin{bmatrix} R_1 & Q_1 \\ \vdots & \vdots \\ R_n & Q_n \end{bmatrix}.$$

Here, $H_4 \in \mathbb{R}^{n \times 2}$.

$$\mathcal{M}' = H_4 \phi. \tag{3.18}$$

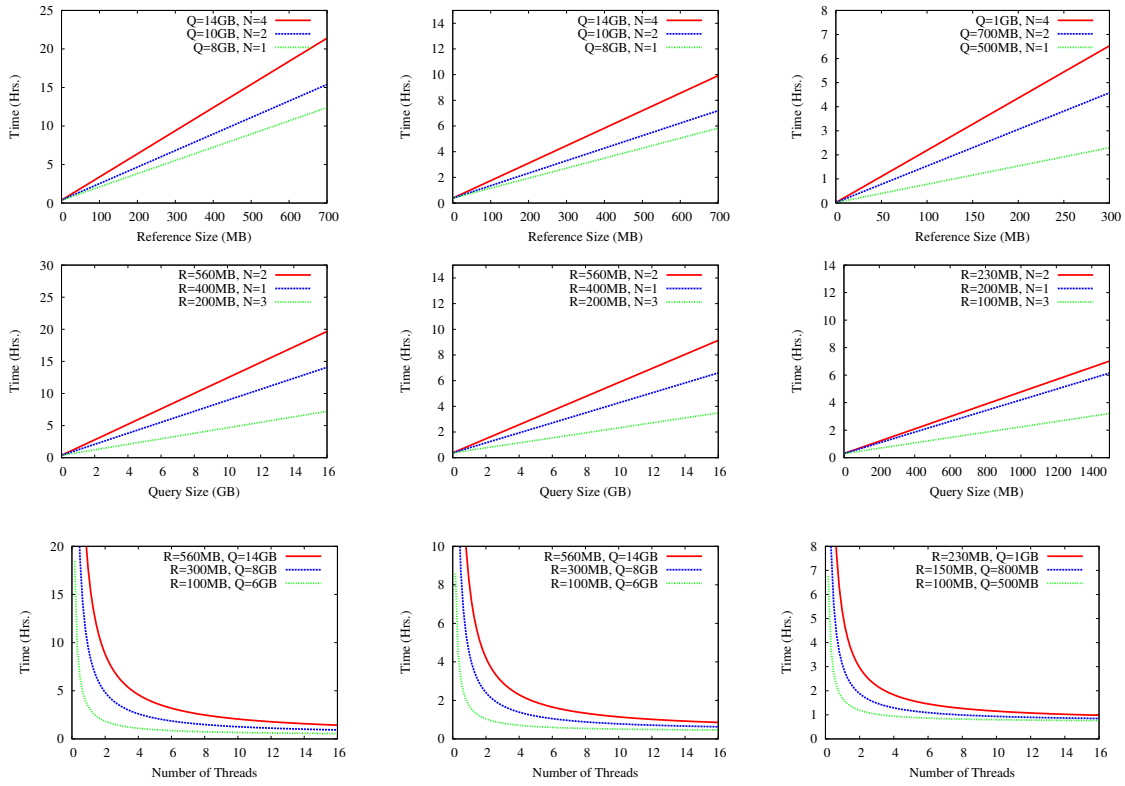
$$\phi^* = H_4^\dagger \mathcal{M}'. \tag{3.19}$$

The memory required by each worker can be computed using equation (3.4).

3.3 Results And Discussion

The optimal coefficients, namely, $\beta^*, \gamma^*, \eta^*$, and ϕ^* are computed using equations (3.3), (3.6), (3.16), and (3.19), respectively. These values are then used to estimate the runtime and memory usage of application-level and system-level models for the test data set. Finally, the estimated values are compared with the empirical

values to measure the accuracy of the models. Figure 3.1 presents model-predicted runtimes of the application-level model for varying configurations of reference size (R), query size (Q), and number of threads (N). Figure 3.2 shows model-predicted memory usage of the application-level model for varying parameters. Similarly, Figure 3.3 and Figure 3.4 demonstrate model-estimated runtime and memory consumption, respectively, of the system-level model for varying number of tasks and cores allocated to each task.



(a) BWA (b) Bowtie2 (c) BLASR

Figure 3.1: Runtimes predicted by application-level model (equation 3.1) for varying sizes of reference (R), query (Q), and number of threads (N) in BWA, Bowtie2, and BLASR. Figures in the first row depict linear behavior of runtime with respect to varying reference size. Figures in the second row show the linear dependence of runtime on the size of query data. Figures in the third row confirm that although runtime reduces with more threads, the corresponding speedup is not proportional, as supported by Amdahl's law [13].

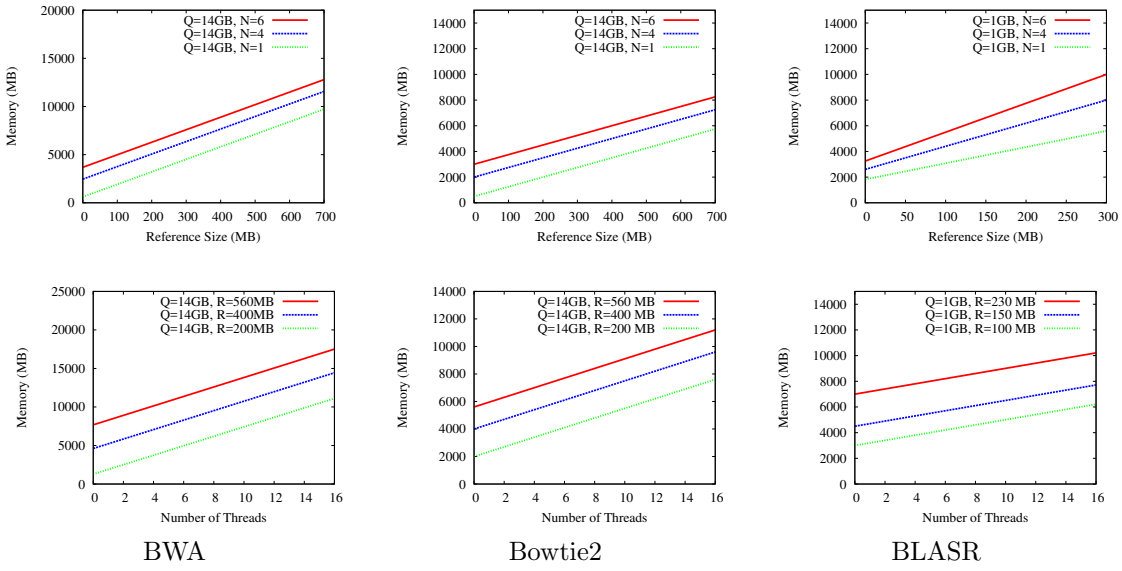


Figure 3.2: Memory usage predicted by application-level model (equation 3.4) for varying reference size (R) and number of threads (N) in BWA, Bowtie2, and BLASR. The memory consumed by the applications is directly proportional to the reference size and number of threads used.

We randomly select 10000 unique subsets (out of $\binom{230}{100}$ possibilities) of size 100 each from the training set of 230 data points. We use as many as 10000 subsets such that, according to the Central Limit Theorem, we get a sufficiently large number of iterations of random variables that follow the normal distribution, as shown later. For each subset, we train the regression models and evaluate coefficients $\beta^*, \gamma^*, \eta^*$, and ϕ^*. At the end of training, we obtain 10000 values for each coefficient. To test the robustness of our proposed approach, given these training data, we present the distribution of the values for each regression coefficient in Figure 3.5. It can be observed that the histograms follow the normal or gaussian distribution where the mean and variance clearly indicate the characteristics of the distribution. The standard deviation (SD) in each case is low, showing that our model is robust. In Table 3.1, we present the Mean Absolute Percentage Error (MAPE) of each model when subjected

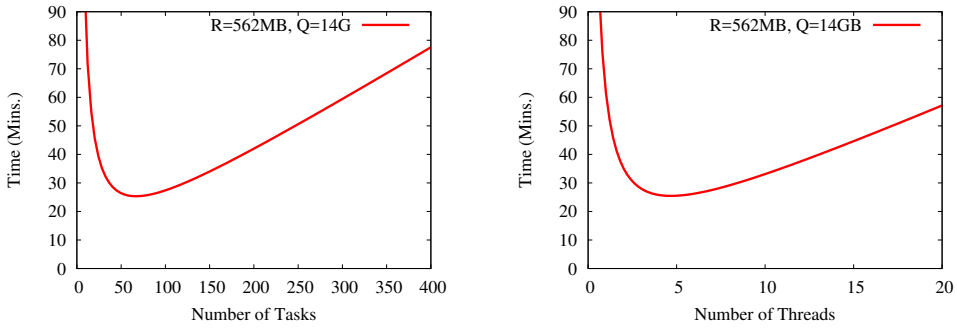


Figure 3.3: Runtimes predicted by system-level model (equation 3.7) for varying number of tasks (K) and threads (N) used by each task. As the number of tasks increases, the runtime gradually decreases unless it reaches an optimal K and N, beyond which the performance is again degraded. This is due to the overhead of splitting, starting up, and joining a given workload.

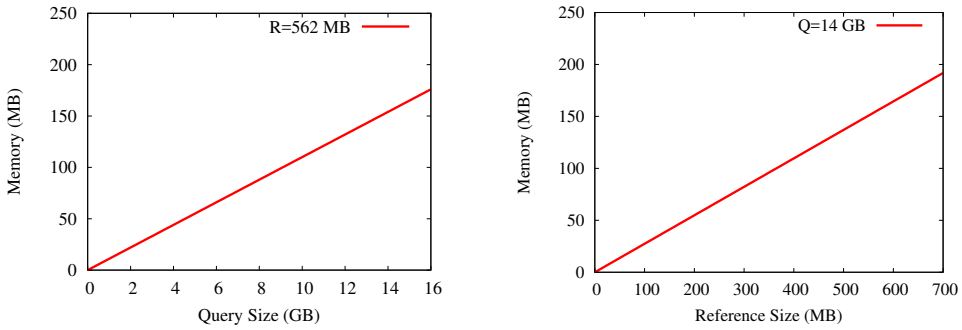


Figure 3.4: Memory usage at the master predicted by system-level model (equation 3.17). The memory footprint of the master server depends on the data to be split and the data to be joined (R and Q).

to different parameter configurations. The low errors indicate that our models can accurately predict runtime and memory when compared to the empirical values of the test set.

3.3.1 Thread-level Parallelism Through Multithreading

The runtime of a computation-heavy workload can reduce with the use of more computing threads. This does not necessarily guarantee a proportionate improvement in speedup. The plots in the last row of Fig. 3.1 show that although initially using more threads reduce runtime, the rate of speedup is not linear. Beyond a certain number, 4 in this case, additional threads do not improve the performance proportionately. This runtime behavior is also supported by Amdahl's law [13], which states that the extent of parallelism is limited by the sequential fraction(s) of the program. We consider this argument while developing the application-level model for runtime estimation. In equation (3.1), the first part ($\frac{RQ}{N}$) denotes the part of the program that can be parallelized with multithreading, whereas the coefficient β_2 signifies the fraction of the program that cannot be parallelized, including the overhead of using multiple threads.

3.3.2 Task-level Parallelism Through Distributed Computing

Data-intensive applications often adopt a distributed implementation wherein data-parallel workloads are split into smaller, independent tasks to be run concurrently. As the number of tasks created for operation increases, the corresponding runtime decreases as long as resources are available to run more tasks. After workload decomposition, the size of data in each task reduces, thereby lowering the execution time of each task. It is important to determine an optimal granularity value, also known as the number of tasks, that would maximize overall performance while

TABLE 3.1

MAPE OF APPLICATION-LEVEL AND SYSTEM-LEVEL MODELS OF RUNTIME AND MEMORY WITH VARYING PARAMETERS

Model	**Application**	**Configuration**		**MAPE(%)**
		Fix	**Vary**	
Application-level - Time	BWA	R	Q,N	3.4
		Q	R,N	3.8
		N	R,Q	2.6
	Bowtie2	R	Q,N	1.6
		Q	R,N	2.2
		N	R,Q	1.3
	BLASR	R	Q,N	4.3
		Q	R,N	5.1
		N	R,Q	3.6
Application-level - Memory	BWA	R	N	3.9
		N	R	3.3
	Bowtie2	R	N	2.6
		N	R	1.9
	BLASR	R	N	4.7
		N	R	4.2
System-level - Time		K	R,Q,P	2.1
		N	R,Q,P	2.7
System-level - Memory		R	Q	2.5
		Q	R	3.3

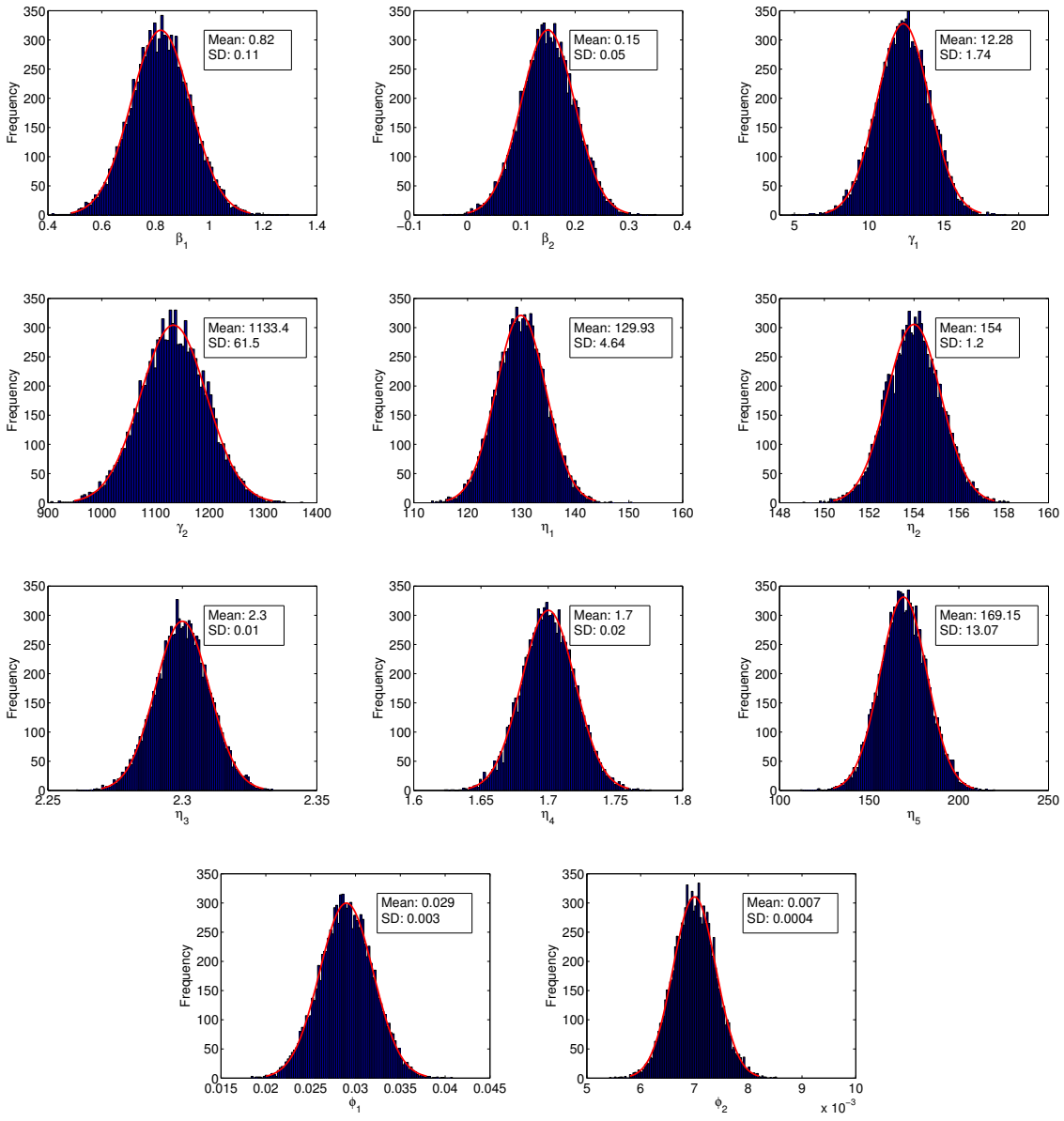


Figure 3.5: Distribution of values of the coefficients in the regression models (equations (3.1), (3.4), (3.13), and (3.17)). Each histogram contains 10000 values of a coefficient obtained while training each model 10000 times. As the distributions follow the gaussian curve, the mean and variance completely characterize the distributions. The calculated SD is low, showing our model is robust for these training data.

factoring in the overhead incurred during splitting the workload, transferring data across the network, and carefully merging the individual outputs. The system-level model discussed in equation (3.13) considers the effect of increasing tasks on system behavior. This can also be seen in Figure 3.3 where beyond a certain sweet spot, the benefit of increased parallelism is outweighed by the underlying overhead.

3.3.3 Balancing Thread-level Parallelism And Task-level Parallelism

While decomposing a workload into multi-core tasks, it is important to find an optimal number of tasks such that maximum parallelism can be attained. It is also crucial to use optimal number of threads such that maximum efficiency can be achieved. As discussed in section 3.3.1, although the use of more threads or cores reduces the overall runtime of an application, the efficiency is not considerably improved. For a given configuration (number of cores and RAM) of computing nodes, there lies a sweet spot for splitting the workload to run on a distributed system. Following the discussion in Section 3.3.2, for a given number of resources P, such that $K_i N_i = P$, where K_i is the number of tasks and N_i is the number of cores used by each task for the i^{th} measurement, we should select optimal values of K and N in order to optimize the overall runtime and inherent costs of splitting tasks, transferring data, and merging outputs. Figure 3.6 illustrates how runtime varies with the number of tasks and the number of threads or cores used by each task, as captured by our model. For the given test workload, using 90 tasks with 4 cores per task is the optimal configuration.

3.3.4 Using Optimal Number Of Computing Instances

For considerably large number of tasks, it might be appealing to use as many computing nodes or instances as possible to reduce completion time. Assuming a single computing instance is associated with each task, if the number of tasks is lower than the number of available instances, all the tasks can be executed in a

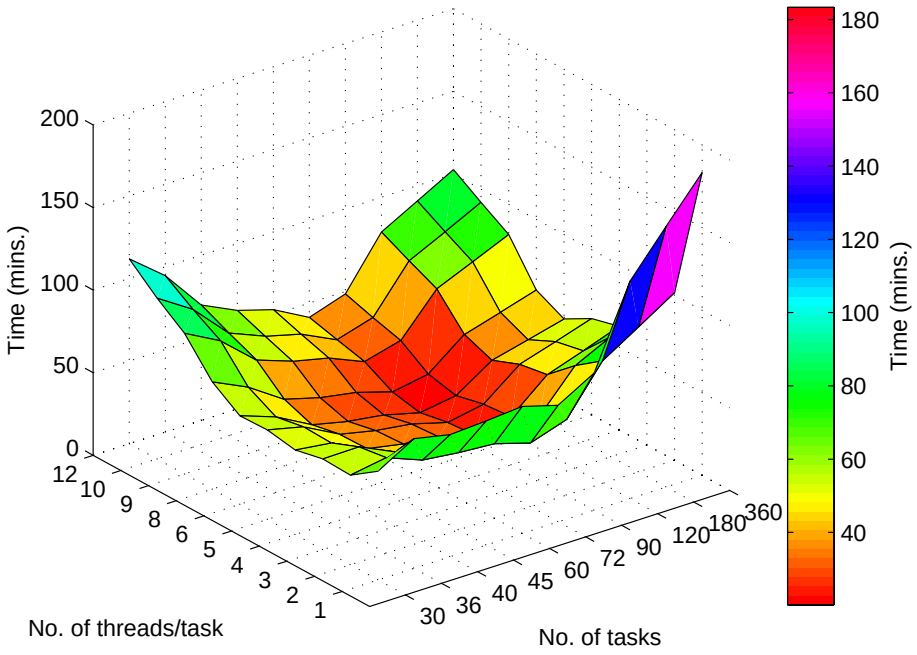


Figure 3.6: Impact of multithreading (thread-level parallelism) and distributed computing (task-level parallelism) on the execution time of a data-intensive workload. Selecting a good runtime configuration can optimize resource utilization. This graph shows that for the test workload, using 90 tasks and 4 cores yields optimal results.

single round after transferring. On the other hand, if the number of tasks is higher than the number of available instances, the workload requires multiple rounds. In such cases the amount of transferred data can be reduced if shared data between tasks is cached, as is possible in our implementation. Thus, data transfer should be considered, specifically in workloads such as cancer sequencing where the reference genome is large and can benefit from caching.

3.3.5 Reducing Cost Of Operation

Many cloud computing services allow users to select resources with differing base cost. Services like Amazon EC2 offers pay-as-you-go scheme where it charges customers based on instance hours used for multiple instances. Customers must ensure

less utilization of resources as well as shorter duration of usage to minimize cost. For instance, although the hourly price of an 8-core instance is higher than a 4-core instance, the speedup obtained from the former may reduce the overall billing usage, and prove to be more cost-efficient. We should also consider the case where customers have to pay for the entire instance hour, although the resources were used for a fraction of it. Hence, the optimization of different parameters governing the operation of tasks directly leads to cost saving. In Table 3.2, we compare the speedup and computation cost for executing the test workload using our model-estimated optimized parameters and set of default parameters. We evaluate costs of harnessing resources from popular utility services like Amazon EC2 and Microsoft Azure. The results show that the optimal configuration (90 tasks and 4 cores) estimated by the model generates maximum speedup and minimum cost of operation.

TABLE 3.2

PERFORMANCE FOR DIFFERENT CONFIGURATIONS BASED ON AMAZON EC2 AND MICROSOFT AZURE-BASED PRICING

Tasks (K)	**Cores (N)**	**MP Time**	**Speedup**	**AEE Cost**	**MAE Cost**
360	1	70	6.6	50.4	64.8
180	2	38	12.3	25.2	32.4
90	4	24	19.5	18.9	32.4
45	8	27	17.3	18.9	32.4

3.4 Conclusion

We address the challenge of optimizing computational resource utilization for executing large workloads on distrubuted systems. To achieve this, we discuss the importance of balancing thread-level parallelism (through multithreading) and task-level parallelism (through distributed computing). We show that although using more cores or threads of execution reduces completion time of a workload, speedup is not proportionately improved. It also incurs higher memory consumption. Decomposing a workload into multiple concurrent tasks seems beneficial due to increased parallelism; however, care should be taken in determining the number of tasks and number of threads allocated to each task. Similarly, utilization of more computing nodes does not guarantee improved performance. Cloud services like Amazon EC2 offers pay-as-you-go scheme, wherein inefficient utilization of computing instances can significantly increase the cost of operation. To mitigate these problems, it is imperative to determine and adopt optimal parameters of operation.

Our proposed predictive models estimate optimal runtime configurations that reduce resource utilization while executing large-scale applications, particularly in the domain of big data genomics. Its hybrid structure consists of an application-specific layer that is incorporated into a generic system-level layout. The models exhibit high predictive capability in determining parameters that optimize runtime, speedup, and cost of using commercial cloud services.

CHAPTER 4

IMPUTATION OF MISSING GENOTYPE DATA IN MODEL AND NON-MODEL ORGANISMS

The following manuscripts describe the work in this chapter.:

- O. Choudhury, A. Chakrabarty, S. J. Emrich. *Highly accurate and efficient data-driven methods for genotype imputation*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, PP(99):1-1, 2017. ISSN 1545-5963. doi: 10.1109/TCBB.2017.2708701.
- O. Choudhury, A. Chakrabarty, S. Emrich. *HAPI-Gen: Highly Accurate Phasing and Imputation of Genotype Data*. ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, 2016.

4.1 Background

High-throughput techniques like whole genome sequencing, whole exome sequencing, and genome-wide single nucleotide polymorphism (SNP) microarrays are generating huge volumes of genotype data. To associate phenotypes, such as disease susceptibility, with underlying genotypes, there has been a rapid growth of phasing-based inference formalisms. Phasing is particularly useful in genome-wide association studies (GWAS) [61] to infer linked alleles on a chromosome. Other downstream analyses include identifying recombinant breakpoints [75], deducing history of human demographics [135], and modeling cis-regulation of gene expression [133].

Missing genotype data is a major hindrance to phasing. This is a result of inherent shortcomings of the underlying techniques that generate such data. Previous efforts have shown that genotype imputation can improve phasing quality in genetic

association studies by up to 10% [129]. Although we focus on imputation as a precursor to phasing, secondary benefits of formulating data-driven imputation methods include generating higher fidelity genetic maps and improved metagenomic analysis (see for example, [49, 142]).

A majority of the existing imputation methods require a panel of reference genotypes and a physical or genetic map. The absence of such a reference panel, as in *non-model* organisms, makes the problem of imputation and phasing much more difficult using currently available software. We address this gap with a lightweight framework, referred to as ADDIT-NM, for fast and accurate imputation in non-model organisms that relies only on the underlying statistics of the genotype data. A preliminary version of ADDIT-NM has been reported in [35]. We also demonstrate that the model organism specific variant of ADDIT, referred to as ADDIT-M, can extract available information in the reference panels via supervised learning to significantly improve imputation accuracy. We perform an extensive, comparative numerical study of ADDIT against the leading imputation tools, such as Beagle [17], IMPUTE2 [63] (for model and non-model organisms) and LinkImpute [112] (for non-model organisms). The comparison results are compiled using real data of varying sizes, varying proportions of missing genotypes, and varying sizes of reference panels (for model organisms). In these comparisons, ADDIT consistently outperforms the other tools in terms of speed, memory, and accuracy.

Our primary contributions include:

- the formulation of data-driven, lightweight imputation algorithms for both model and non-model organisms with high speed and accuracy;
- the incorporation of both local and global information by utilizing adaptive windows and trust metrics;
- the exploitation of adjacent genotype data to significantly expedite imputation under certain conditions;
- the employment of multi-class supervised learning algorithms to extricate information from reference panels of model organisms to enhance the imputation

process.

4.2 Methods

In this section we present a detailed description of our proposed ADDIT algorithm. For ADDIT-NM, the imputation framework is segregated into multiple steps (Steps 1–5), and justification for each step is provided in section 4.2.1. Note that the algorithm in Section 4.2.1 is described in detail in [35]. For ADDIT-M, we present a windowed, multi-class supervised learning-based imputation algorithm in section 4.2.2.

Let N be the number of samples in a given population and M_0 be the total number of missing genotypes in the entire population. Let G_q^j denote the genotype at the jth position of the qth sample. Here, $q \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, M\}$. Let $\mathcal{I}_q$ be the set of potential imputable genotypes at G_q^j. We denote a window centered at G_q^j by $\mathcal{W}(G_q^j, d)$, where d is the window length and the window contains $(d-1)/2$ elements on either side of the central element G_q^j. We use Hamming distance to measure the similarity between two windows of identical length. For example, the Hamming distance between '111000' and '100010' is three. Let $|A|$ represent the cardinality of a set A. A list of important notation/symbols used in the subsequent discussion can be found in Table 4.1.

4.2.1 ADDIT-NM: Imputation For Non-Model Organisms

For non-model organisms, ADDIT-NM uses adaptive windows and likelihoods to estimate missing values. An overall schematic of ADDIT-NM is provided in Figure 4.1.

TABLE 4.1

LIST OF SYMBOLS USED IN THE ADDIT-NM ALGORITHM DESCRIPTION

Symbol	*Meaning*
$\#$	number of
N	# samples in given population
M_0	total # missing genotypes
M	# missing genotypes with distinct neighbors
q	query sample
G_k^j	genotype at jth position of kth sample
$\mathcal{W}(c,d)$	window of length d centered at c
$\mathcal{C}_q$	set of candidate windows for query window q
$\{\mathcal{C}_q\}_{q=1}^M$	family of $\mathcal{C}_q$ for all M query windows
ρ_H	Hamming distance
θ	similarity score
$\theta_{\min}$	similarity threshold
$\mathcal{T}_q$	set of trusted candidates for query window q
$\{\mathcal{T}_q\}_{q=1}^M$	family of $\mathcal{T}_q$ for all M query windows
ω_f	decision weight based on allele frequency
ω_s	decision weight based on window similarity
$\mathcal{I}_q$	set of imputable genotypes for query sample q
i_q	element of $\mathcal{I}_q$
$\mathcal{P}(i_q)$	priority level of i_q according to Table 4.2

4.2.1.1 Step 1: Quick Imputation Using Immediate Neighbors

Suppose G_q^j is a missing genotype. We begin by performing a quick impute (QI) step by imputing the central value to either of its two neighboring alleles if $G_q^{j-1} = G_q^{j+1}$; this is described in Algorithm 1 and illustrated in Figure 4.1A. The QI step is

Algorithm 1 Quick imputation using immediate neighbors

Require: Genotype data with M_0 missing values
Require: Window length, d
1: **for** each missing G_q^j **do**
2: **if** $G_q^{j-1} = G_q^{j+1}$ **then**
3: Quick impute $G_q^j \leftarrow G_q^{j+1}$
4: **end if**
5: **end for**

effective in substantially reducing the search space during imputation, if the likelihood of double recombination in adjacent alleles is low.

4.2.1.2 Step 2: Similarity Computation For Each Missing Genotype

For a missing value G_q^j in the query sample, we create a **query window** of length d, denoted $\mathcal{W}(G_q^j, d)$. We construct windows of identical length centered at the jth position of each of the remaining population samples, which we call **sample windows**, denoted $\mathcal{W}(G_k^j, d)$, where $k \in \{1, \ldots, q-1, q+1, \ldots, N\}$.

We now exclude those sample windows that have missing data at G_k^j. The subset of sample windows with no missing data at G_k^j is hereby referred to as the **candidate set** $\mathcal{C}_q$ for the missing genotype G_q^j (Figure 4.1B). Each sample window is given a **similarity score** using

$$\theta_i = d - \rho_H \left(\mathcal{W}(G_q^j, d), \mathcal{W}(G_i^j, d) \right) \tag{4.1}$$

Algorithm 2 Similarity computation for each missing genotype

Require: M missing genotypes with distinct neighbors

for each missing G_q^j with distinct neighbors **do**
  Construct query window $\mathcal{W}(G_q^j, d)$
  for each $k \in \{1, \ldots, N\}$ except q **do**
    if G_k^j is not missing data **then**
      $\mathcal{C}_q \leftarrow$ add $\mathcal{W}(G_k^j, d)$ to the set of candidate windows
    end if
  end for
  for $i =$ each candidate window in $\mathcal{C}_q$ **do**
    $\theta_i \leftarrow$ similarity score using (4.1)
  end for
end for
return set of candidate windows for M missing genotypes, $\{\mathcal{C}_q\}_{q=1}^{M}$
return set of similarity scores for all candidate windows, $\{\theta_q\}_{q=1}^{M}$

where ρ_H is the Hamming distance between the query window and the ith sample window. Iterating over the remaining set of missing genotypes after the QI step, we acquire a set of candidate windows, $\{\mathcal{C}_q\}_{q=1}^{M}$ with a corresponding set of similarity scores $\{\theta_q\}_{q=1}^{M}$.

4.2.1.3 Step 3: Similarity Threshold Of Candidate Windows

Consider a histogram of similarity scores for the collection of candidate windows denoted as $\mathcal{H}(\theta)$. A maximum likelihood

$$\theta_{\min} = \arg\max_{\theta} \mathcal{H}(\theta) \tag{4.2}$$

is used to compute a **similarity threshold**, the procedure for which is provided in Algorithm 3.

It is important to note that the value of $\theta_{\min}$ is computed considering all candidate windows of the M missing genotypes remaining after Step 1. The inherent globality in our formulation offers various advantages: First, it avoids bias induced by local candidate windows with low similarity scores. For example, suppose that

Algorithm 3 Similarity threshold of candidate windows

Require: $\{\mathcal{C}_q\}_{q=1}^{M}, \{\theta_q\}_{q=1}^{M}$ from Algorithm 2

1: **for** $r = 1$ to $d - 1$ **do**

2: $\mathcal{H}(\theta) \leftarrow$ frequency of windows in all candidate sets $\{\mathcal{C}_q\}_{q=1}^{M}$ with similarity score r

3: **end for**

4: $\theta_{\min} \leftarrow \arg\max_{\theta} \mathcal{H}(\theta)$

5: **return** Similarity threshold, $\theta_{\min}$

local candidate windows for a particular missing genotype have similarity scores between 2 and 7, with window length $d = 11$. Also consider that the entire sample population contains windows (with identical d) of similarity scores between 2 and 10 with most windows having scores > 6. If we compute $\theta_{\min}$ based on local candidates only, then $\theta_{\min}^{\text{local}}$ is (say) 4. However, using the globally-derived similarity threshold $\theta_{\min}$, we obtain $\theta_{\min}^{\text{global}} = 6$ because the global sample population has more candidate windows with higher similarity scores. If we were to impute a value based on $\theta_{\min}^{\text{local}}$, then we would enable windows with lower similarity scores to have an effect on the decision, which is undesirable, as it is likely to recommend an erroneous imputed value. Instead, using $\theta_{\min}^{\text{global}}$ filters out these low-similarity candidates, resulting in more accurate imputation. Finally, our method avoids diluting information; using candidate windows retains relevant local information with embedded global trends of these data. This improves imputation accuracy while lowering required computation.

We will next leverage the notion of the similarity threshold to categorize candidate windows as trusted or untrusted.

4.2.1.4 Step 4: Adaptive Classification Of Trusted Candidates

For the qth query window we construct a set of **trusted candidates**, denoted $\mathcal{T}_q$. A candidate window is said to be a trusted candidate if its similarity score is at least the similarity threshold $\theta_{\min}$ (as shown in Figure 4.1C). Note that $\mathcal{T}_q$ cannot be empty for the choice of $\theta_{\min}$ in (4.2). This claim can be proven by contradiction, given that

there is at least one candidate window for a given window length $d > 2$. Suppose $\mathcal{T}_q$ is empty for a given $\theta_{\min} := \arg\max_\theta \mathcal{H}(\theta)$. This implies that there is no candidate window whose similarity score is at least $\theta_{\min}$. Clearly, this is a contradiction because the set of candidate windows is not empty, so at least one candidate window must have similarity score $\theta_{\min}$ in order to ensure that it is the maximizer of the histogram $\mathcal{H}(\theta)$. Therefore, $\mathcal{T}_q$ must be non-empty for this choice of $\theta_{\min}$.

We refer to this method as *adaptive* because it allows each $\mathcal{T}_q$ to have a variable number of trusted candidate windows, unlike existing frameworks such as those employing k-nearest neighbor algorithms. This is advantageous because it exploits only the most similar windows for subsequent imputation. To take into account both window similarity and repetitiveness of the central allele, we introduce the following priority-based weighting scheme.

Algorithm 4 Adaptive classification of trusted candidates

Require: Set of candidates $\{\mathcal{C}_q\}_{q=1}^M$ with similarity values $\{\theta_q\}_{q=1}^M$, obtained in Algorithm 2; similarity threshold, $\theta_{\min}$

```
for each missing genotype G_q^j do
    C_q ← set of candidate windows for G_q^j
    for each candidate window in C_q do
        θ ← similarity score of candidate window
        if θ ≥ θ_min then
            T_q ← add candidate window to the set of trusted candidates
        end if
    end for
end for
return All M sets of trusted candidates, {T_q}_{q=1}^M
```

4.2.1.5 Step 5: Priority-based Imputation Scheme

Recall that $\mathcal{I}_q$ is the set of potential imputable genotypes at G_q^j. For each imputable genotype $i_q \in \mathcal{I}_q$, we assign **decision weights** based on two criteria: (i) the frequency of i_q at the central element over all trusted candidate windows in $\mathcal{T}_q$; and (ii) the similarity score of trusted candidate windows containing i_q in the central position (refer to Figure 4.1D). The window similarity decision weight ω_s indicates the reliability of $T_q^{i_q}$ for imputing the missing genotype with i_q. The allele frequency decision weight ω_f signifies the likelihood of i_q, even if the corresponding trusted candidates have low similarity scores with respect to the query window. The decision weights are designed to handle potential bias towards highly frequent genotypes found in trusted candidates with low similarity scores. Mathematically, the decision weights are written as:

$$\omega_f(i_q) = \frac{F_{i_q}}{|\mathcal{T}_q|}, \tag{4.3a}$$

$$\omega_s(i_q) = \frac{1}{F_{i_q}} \sum_{k \in \mathcal{T}_q^{i_q}} \frac{d - \rho_H\left(\mathcal{W}(G_q^j, d), \mathcal{W}(G_k^j, d)\right)}{d-1}, \tag{4.3b}$$

where F_{i_q} is the frequency of i_q at the central position of the trusted candidates, $\mathcal{T}_q^{i_q} \subset \mathcal{T}_q$ is the set of trusted candidates with i_q in the central position, and $\rho_H\left(\mathcal{W}(G_q^j, d), \mathcal{W}(G_k^j, d)\right)$ is the Hamming distance between the query window and each window $\mathcal{W}(G_k^j, d) \in \mathcal{T}_q^{i_q}$.

We categorize the values of $\omega_f(i_q)$ and $\omega_s(i_q)$ as high, medium, or low. For this categorization, we use data in the histogram obtained in Step 3 as follows. We first eliminate all windows with similarity scores below $\theta_{\min}$ as in Step 3. Thus, the lowest allowable similarity score is $\theta_{\min}$, which motivates us to classify $< \theta_{\min}/d$ as low. Let $\theta_{\max}$ be the highest similarity score on the histogram with non-zero frequency. Then

we classify high as $> \theta_{\max}/d$. Note that $\theta_{\max}/d < 1$ since the maximal similarity is $d-1$. All decision weights in the range $[\theta_{\min}/d, \theta_{\max}/d]$ are considered medium. For each imputable genotype i_q, we determine its priority level $\mathcal{P}(i_q)$ using the rules in Table 4.2.

Within Table 4.2, the priorities are set such that higher weights are given to imputable genotypes supported by highly similar trusted candidates irrespective of ω_f. This is motivated by the fact that, in haplotypes, we expect highly similar samples over local regions to exhibit identical inheritance of genotypes. If the trusted candidates have medium similarity, then we check ω_f. This is because the trusted candidates cannot be completely relied upon to generate a correct imputed genotype. Instead, we also rely on a likelihood-based estimate embedded into ω_f. For an imputable genotype with medium ω_s, if its corresponding ω_f is high, then that holds more priority than medium or low ω_f. We will assign even less priority to the genotypes that have high or medium frequencies with low ω_s for similar reasons as discussed above. Finally, we will assign the least priority to a genotype if its supporting trusted candidates are of low similarity and low frequency. For such cases, we investigate the remaining genotypes in $\mathcal{I}_q$.

We impute the genotype i_q with the highest priority level. If there is a clash of priorities, either value can be imputed. This is written mathematically as:

$$\hat{i}_q = \arg\min_{i_q \in \mathcal{I}_q} \mathcal{P}(i_q), \tag{4.4}$$

where $\hat{i}_q$ is the imputed genotype. The pseudo-code for this step is presented in Algorithm 5.

TABLE 4.2

PRIORITY LEVEL FOR POSSIBLE COMBINATIONS OF DECISION WEIGHTS DURING STEP 5 OF ADDIT-NM

ω_s	ω_f	Priority based on	$\mathcal{P}$
High	-	Window Similarity	1
Med	High	Allele Frequency	2
Med	Med/Low	Window Similarity	3
Low	High/Med	Allele Frequency	4
Low	Low	-	5

Algorithm 5 Priority-based imputation scheme

Require: Set of trusted candidates $\mathcal{T}_q$ for query window $\mathcal{W}(G_q^j, d)$
Require: Set of imputable genotypes $\mathcal{I}_q$
1: **for** each imputable genotype $i_q \in \mathcal{I}_q$ **do**
2: $\quad \mathcal{T}_q^{i_q} \leftarrow$ trusted candidates with i_q in the central position
3: $\quad \omega_f(i_q),\ \omega_s(i_q) \leftarrow$ decision weights using eqn. (4.3)
4: $\quad$ Categorize ω_f and ω_s as 'high', 'medium', or 'low'
5: $\quad \mathcal{P}(i_q) \leftarrow$ priority level according to Table 4.2
6: **end for**
7: $\hat{i}_q \leftarrow$ using eqn. (4.4)
8: **return** Imputation decision, $\hat{i}_q$

4.2.2 ADDIT-M: Imputation For Model Organisms

4.2.2.1 Step 1: Construction Of Training And Truth Sets From Reference Panel

Let N_{train} be the number of training samples collected to train a supervised learning machine $\mathcal{L}$, and R_k^j denote the jth position of the kth reference sample. Recall that G_q^j is the missing value in the query sample at the jth position, and d is a positive integer that denotes the number of features of the training set.

For each G_q^j, we begin by constructing a truth set, $\mathcal{S}_{\text{truth}}^j \in \mathbb{R}^{N_{\text{train}}}$, and training

set, $\mathcal{S}^j_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times (d-1)}$: these will be used by a classifier $\mathcal{L}$ to inform the imputation process. The truth set $\mathcal{S}^j_{\text{truth}}$ is constructed using N_{train} genotypes from the reference panel that do not have missing data at the jth position, that is:

$$\mathcal{S}^j_{\text{truth}} = \begin{bmatrix} R^j_1 & R^j_2 & \cdots & R^j_{N_{\text{train}}} \end{bmatrix}.$$

The training set construction (feature selection) is more involved. One cannot select a training set containing reference data with indices belonging to a window of length d centered at R^j_k for $k = 1, 2, \ldots, N_{\text{train}}$, because the corresponding testing set (a window of length d centered at G^j_q) could contain missing data, which may result in low-quality predictions. Instead the training set is selected from the reference panels corresponding to indices in the neighborhood of G^j_q that do not contain missing values. This can be written more rigorously as

$$\mathcal{S}^j_{\text{train}} = \begin{bmatrix} R^{m_1}_1 & R^{m_2}_1 & \cdots & R^{m_{d-1}}_1 \\ R^{m_1}_2 & R^{m_2}_2 & \cdots & R^{m_{d-1}}_2 \\ \vdots & \vdots & \ddots & \vdots \\ R^{m_1}_{N_{\text{train}}} & R^{m_2}_{N_{\text{train}}} & \cdots & R^{m_{d-1}}_{N_{\text{train}}} \end{bmatrix},$$

where $\{m_i\}_{i=1}^{d-1}$ is a set of indices representing $d-1$ nearest neighbors to G^j_q containing no missing values. The corresponding test set is given by

$$\mathcal{S}^j_{\text{test}} = \begin{bmatrix} G^{m_1}_q & G^{m_2}_q & \cdots & G^{m_{d-1}}_q \end{bmatrix}.$$

The formation of the training, testing, and truth set is illustrated in Figure 4.2A.

4.2.2.2 Step 2: Imputation Based On Identical Truth Values

Clearly, if all the labels in the truth set $\mathcal{S}^j_{\text{truth}}$ are identical, there is no need to train the classifier $\mathcal{L}$. In such a case, the imputed value is the label in $\mathcal{S}^j_{\text{truth}}$.

4.2.2.3 Step 3: Quick Imputation

This is an optional step, which performs effectively for data exhibiting a low degree of double recombination in adjacent positions. The implementation of this step has been previously discussed in Step 1 of Section 4.2.1.

4.2.2.4 Step 4: Imputation Via Multi-class Supervised Learning

If the conditions in the earlier steps are not satisfied, this implies that the truth set $\mathcal{S}^j_{\text{truth}}$ contains more than one unique label. In fact, $\mathcal{S}^j_{\text{truth}}$ could contain multiple labels; for example, three labels if the genotypes are encoded with $\{0, 1, 2\}$. The learning machine $\mathcal{L}$ is, therefore, referred to as a multi-class learning machine [11]. The multi-class classifier $\mathcal{L}$ learns from the training set $\mathcal{S}^j_{\text{train}}$ and the corresponding multi-class truth values $\mathcal{S}^j_{\text{truth}}$, and can consequently be used to predict the value of G^j_q using the test set $\mathcal{S}^j_{\text{test}}$. This procedure is illustrated in Figure 4.2B–C.

4.3 Results And Discussion

4.3.1 Testing ADDIT-NM

4.3.1.1 Data Acquisition

We test ADDIT-NM on three benchmark plant datasets considered in [112] (see Table 4.3 for details). In summary, we use genotype by sequencing (GBS) data from members of the grape genus *Vitis* generated by Illumina Hi-Seq and mapped to its reference genome [6, 68]. Some SNPs based on missing values, heterozygosity,

TABLE 4.3

COMPARISON OF THE PERFORMANCE OF ADDIT-NM WITH BEAGLE, LINKIMPUTE, AND IMPUTE2

Dataset	**# Samples (N)**	**# SNPs**	**# Missing Genotypes (M_0)**	**Method**	**Error (%)**	**Runtime (s)**	**Memory (MB)**
Grape	77	8506	2000	Beagle	11.0	16	371
				LinkImpute	9.5	28	3465
				IMPUTE2	13.4	18	19
				ADDIT-NM	**2.6**	**14**	**7**
Apple	711	8404	10000	Beagle	7.6	424	804
				LinkImpute	7.4	104	6941
				IMPUTE2	9.2	98	45
				ADDIT-NM	**5.3**	**86**	**17**
Maize	4300	43695	10000	Beagle	18.7	16585	927
				LinkImpute	18.1	7608	11333
				IMPUTE2	21.4	7492	686
				ADDIT-NM	**8.7**	**7233**	**378**

and minor allele frequency (MAF) as in [112] are discarded. A similar apple dataset generated from members of the genus *Malus* is acquired from the 1995 accession from the US Department of Agriculture repository in Geneva, NY. The samples are double-digested with restriction enzymes and sequenced with Illumina Hi-Seq. The reads are mapped to the reference genome of *Malus domestica* version 1.0 [138]. Similar to the above grape data, variants are also filtered. Finally, we consider a large maize (corn) dataset available at the International Maize and Wheat Improvement Center [60] to verify the scalability of our proposed algorithm. For this dataset, a pre-processing stage eliminates bi-allelic SNPs with $< 20\%$ missing data, minor allele frequencies (MAF) of $> 1\%$, and samples with $> 20\%$ missing values.

4.3.1.2 Comparative Analysis

We implement ADDIT-NM and compare its performance with contemporary imputation algorithms such as Beagle 3.3.2, LinkImpute, and IMPUTE2. Performance metrics used for this comparison include (i) percentage of genotype imputation errors; (ii) runtime; and (iii) memory usage. The results of this comparative study are tabulated in Table 4.3. It is clear that ADDIT-NM significantly outperforms the competition. For example, the genotype errors of grape and maize imputation are less than half the minimum of the errors produced by the other methods. The runtime of ADDIT-NM is consistently small, at times an order of magnitude smaller than the corresponding runtimes of Beagle and/or LinkImpute. Importantly, this large speed-up does not result in prohibitive use of memory. This is demonstrated by a 2–3 order-of-magnitude reduction of memory usage in comparison with Beagle or LinkImpute, and significantly less memory (around half or less) as IMPUTE2.

As discussed in [18], Beagle is more accurate than IMPUTE2 for large sample sizes. IMPUTE2 implements pre-phasing, wherein genotypes are first phased and then haplotypes are imputed. This reduces runtime and memory usage at the cost of accuracy. LinkImpute requires similar runtime as Beagle, although it has slightly higher accuracy. It incurs high computational overhead since it uses a genome-wide similarity search based on k-nearest neighbor imputation (kNNi) [137]. Contrary to these, ADDIT-NM relies on an adaptive number of reliable trusted candidate windows, which helps in increasing imputation accuracy. It also can significantly reduce runtime and memory use via an initial pruning of the search space that we call quick imputation. Unlike Beagle and IMPUTE2, we do not require a large genotype panel, which further reduces the lookup time and memory required and makes ADDIT-NM applicable to less studied organisms.

4.3.1.3 Effectiveness Of Quick Imputation

A major reason for the computational efficiency of ADDIT-NM is due to the quick imputation step. To illustrate the performance of each imputation stage (that is, quick versus priority-based imputation), we refer the reader to Figure 4.3. We observe that the number of quick imputes (QI) is significant for each dataset. However, the corresponding number of quick impute errors (QI Error) are small. For the grape, apple, and maize datasets, 1 out of 1487 ($< 0.1\%$), 210 out of 6326 ($< 4\%$), and 168 out of 6078 ($< 3\%$) genotypes, respectively, are incorrectly imputed in the QI stage. Figure 4.3 also contains information regarding the number of priority imputations (PIs) and their corresponding imputation errors (PI Error). For the real datasets, the proportion of PI Errors is low, ranging from $< 10\%$ in apple and grape, to $< 17\%$ in maize. This trend suggests that for these plant data, the QI step can be exploited because it combines high computational speeds along with a relatively lower imputation error rate.

TABLE 4.4

COMPARISON OF ADDIT-NM WITH AND WITHOUT QI

Dataset	**Error (%)**		**Runtime (min)**		**Memory (MB)**	
	QI	No QI	QI	No QI	QI	No QI
Grape	2.6	3.0	0.2	1.1	7	7
Apple	5.3	5.5	1.4	4.9	17	17
Maize	8.7	9.0	120.6	417.0	378	378

This is further supported by comparing imputation performance of ADDIT-NM with and without the QI step (Table 4.4). We observe that the maximum memory used for both the configurations are identical for all datasets, and the error percentage is comparable; a minuscule increase in the number of errors is noted when the quick impute step is skipped. The most noteworthy result obtained from this investigation is the runtime differences: the lack of the quick impute step results in higher execution time for each dataset, particularly for larger datasets, as expected.

4.3.2 Testing ADDIT-M

We also test the performance of ADDIT-M on human model organism data. We use the multi-class support vector machine (MC-SVM) as an exemplar supervised learning algorithm. The MC-SVM is implemented via Python's `scikit-learn` module. The rationale behind choosing the SVM as our supervised learning method is its ability to handle high-dimensional data using the kernel-trick, its efficiency with smaller-sized training sets [65], and its effectiveness in the imputation problem, as reported in the comparative study [110].

4.3.2.1 Data Acquisition

For testing ADDIT-M, we obtain genotype data of phase 3 human chromosome 20 from the 1000 Genomes Project [5]. This data comprises 2504 individuals from 26 populations. We select a subset comprising 8 populations (GBR, TSI, CHS, STU, GIH, LWK, CHB, IT) and randomly mask 1%, 2%, and 5% of the data for subsequent imputation. We further use 75%, 90%, and 95% of the remaining phase 3 data as reference panels for running Beagle and IMPUTE2 and as the training set for the learning algorithm of ADDIT-M.

TABLE 4.5

COMPARISON WITH BEAGLE ON HUMAN DATA

(A) Fixed Training Set (75%)

Missing (%)	**Method**	**Error (%)**	**Runtime(s)**	**Mem(GB)**
1	Beagle	6.6	1900	3.8
	ADDIT-M	1.0	58.7	1.0
2	Beagle	8.1	2040	3.8
	ADDIT-M	2.5	64.5	1.0
5	Beagle	11.0	2080	3.8
	ADDIT-M	2.8	88.7	1.0

(B) Fixed Missing Genotypes (5%)

Training Size (%)	**Method**	**Error (%)**	**Runtime(s)**	**Mem(GB)**
95	Beagle	2.6	2200	3.5
	ADDIT-M	1.9	94.1	1.1
90	Beagle	5.1	2160	3.8
	ADDIT-M	2.6	91.7	1.1
75	Beagle	11.0	2080	3.8
	ADDIT-M	2.8	88.7	1.0

4.3.2.2 Comparative Analysis

To test our proposed ADDIT-M imputation algorithm, we again consider the overall imputation error percent, total runtime, and maximum memory used. For our experiments, since IMPUTE2 required at least 130× computation time higher than Beagle (also shown in [17]), we only consider Beagle for comparison with the now optional QI step of ADDIT-M (see previous section) turned off.

The results of our comparative study is tabulated in Table 4.5(A) and (B). In Table 4.5(A), we demonstrate the effect of increasing the proportion of missing values. We randomly fix 75% of the data as the reference panel containing no missing data, and mask the remaining data by 1%, 2%, and 5%. We note that ADDIT-M consistently requires less memory (about 1/4th) and demonstrates speedups of two orders of magnitude. Additionally, the overall imputation error percent is considerably lower for ADDIT-M. In Table 4.5(B), we demonstrate results for 5% missing data when the training set size varies amongst 95%, 90% and 75%. As expected, decreasing the number of training samples worsens the performance of the supervised learning algorithm: the total error percent for ADDIT-M gradually increases from 1.9% to 2.8%. Note that the error percent of Beagle exhibits a more accelerated increase with reduction of training size relative to our proposed approach.

4.3.2.3 When Should We Use QI?

As mentioned before, the effectiveness of QI is most pronounced when the adjacent alleles exhibit a low degree of double recombination. Although ADDIT-M with QI completes roughly 7% faster on the human data obtained from [5], it does perform worse (see Table 4.6) using a 75% training set and 5% missing data. In Figure 4.4, we illustrate the distribution of imputation errors over the three decision-making steps of ADDIT-M: the identical truth value (Step 2), QI (optional Step 3), and supervised learning based imputation (Step 4). Since we have a high level of trust

in the reference genotype panel, we give Step 2 the highest priority in terms of determining the imputed value. Thus, the percentage of values imputed in Step 2 remains unaltered with and without QI. It is clear from the figure that the QI step only affects the other 70% of the missing values: specifically 17% of the missing genotypes are eligible for QI. Of these 17%, 10% are imputed incorrectly. The SVM performance, both with and without QI, are very similar and exhibit 4% imputation error (this is because the training samples are identical for both runs). It follows that for these data QI performs relatively worse as compared to using available reference data. Thus we do not recommend the QI step unless adjacent alleles exhibit low degrees of double recombination.

4.3.2.4 Importance Of Multi-class Supervised Learning

We believed that using a supervised learning algorithm would enhance imputation accuracy. As a result, we expect that Beagle and ADDIT-M will outperform ADDIT-NM by exploiting the information embedded in the reference genotype panel. This is indeed the case, as deduced from Table 4.7. Among the two imputation tools for model organisms, ADDIT-M outperformed Beagle in terms of imputation accuracy, runtime, and memory. A considerable subset of the query data was filtered for identical truth (IT) and quick impute (QI)-based deduction, that lead to accurate and expedited imputation. For the remaining set of missing values, the supervised learning approach enabled accurate imputations. The results presented in Table 4.7 show that for model organisms, utilizing genotype information in reference panel, as in the case of Beagle and ADDIT-M, provide more accurate imputations.

Hence, we present ADDIT, the first algorithm for genotype imputation in model and non-model organisms, based on accurate and efficient window-based data-driven approaches. We test our proposed methods on real data sets of humans and plants and demonstrate that for varying sizes of data and training samples, proportions

of missing genotypes, ADDIT consistently outperforms leading tools like Beagle, IMPUTE2, and LinkImpute.

TABLE 4.6

PERFORMANCE WITH AND WITHOUT THE QUICK IMPUTE STEP FOR ADDIT-M ON HUMAN DATA

Dataset	**Error (%)**		**Runtime (s)**		**Memory (GB)**	
	QI	No QI	QI	No QI	QI	No QI
Human	3.8	2.8	84.1	90.4	1.0	1.0

TABLE 4.7

PERFORMANCE COMPARISON FOR BEAGLE, ADDIT-M, AND ADDIT-NM ON HUMAN DATA

Tools	**Error (%)**	**Runtime (s)**	**Memory (GB)**
Beagle	11.0	2080	3.8
ADDIT-M	2.8	90	1.3
ADDIT-NM	14.6	1064	0.06

4.4 Conclusion

Genotype imputation is an essential precursor for improving the quality of haplotype phasing in applications like genome-wide association studies. Although model organisms can resort to available reference genotype panel for imputation, the problem becomes more challenging for non-model organisms that lack such reference data. Here, we present ADDIT, an accurate and efficient window-based data-driven approaches for imputation of missing genotypes in both model and non-model organisms. We test our proposed methods on real datasets of non-model and model organisms, including humans. For varying sizes of data, proportions of missing genotypes, and sizes of training samples, our method consistently performs better than the leading tools like Beagle, IMPUTE2, and LinkImpute.

Although the multiclass classifier approach used in ADDIT-M generated accurate imputations, there still remains a scope to investigate other data-driven supervised learning approaches in Step 4 of section 4.2.2. One can further analyze the performance of our imputation tools when plugged in to different phasing algorithms. Finally, a natural extension of genotype imputation is the devlopment of an accurate haplotype phasing mechanism for downstream analysis. In this regard, one can employ a graph-based phasing approach [132] or further explore sophisticated hidden Markov model-based algorithms.

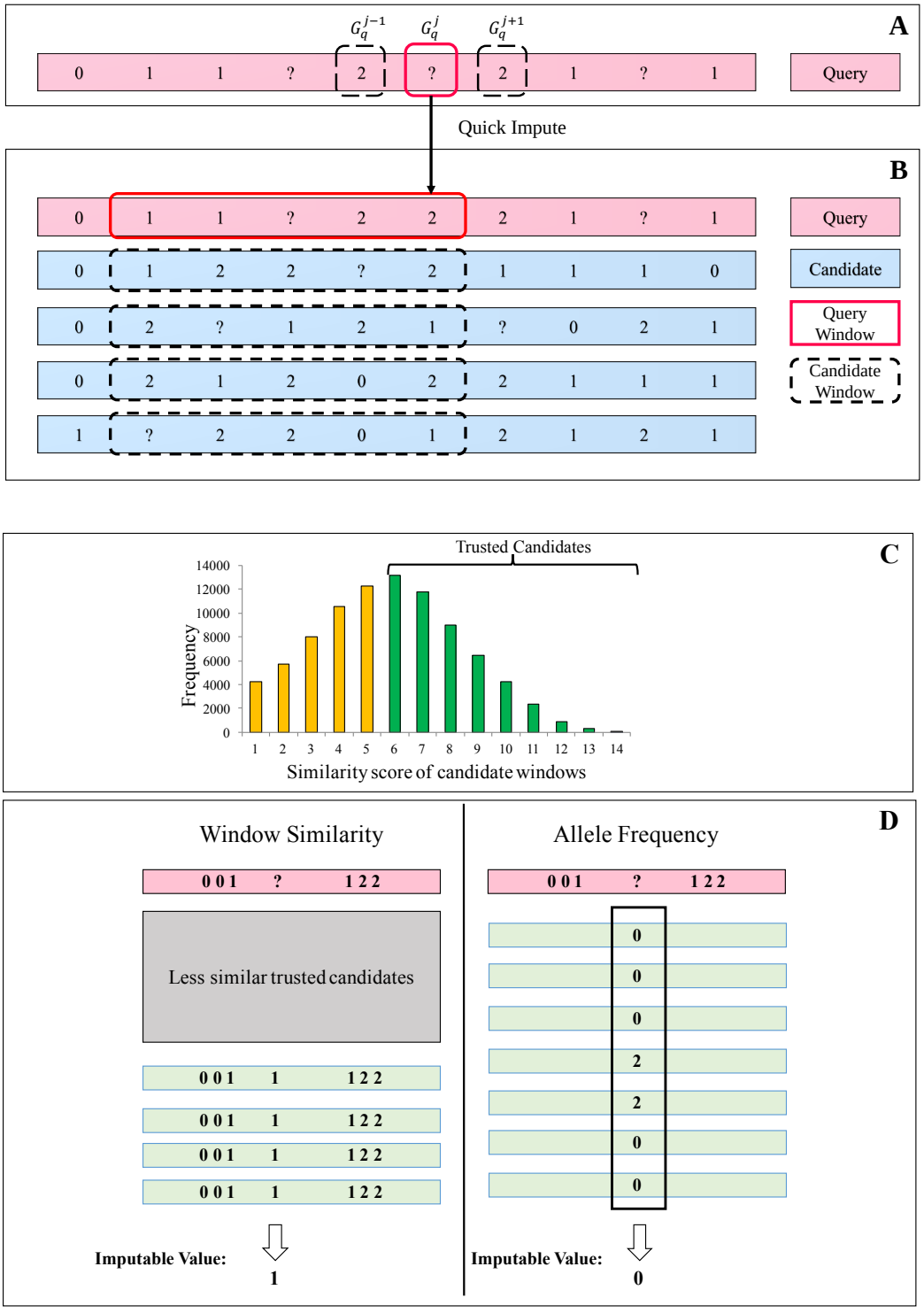


Figure 4.1: ADDIT-NM for non-model organisms: (A) quick impute (QI) step. (B) Selection of candidate windows with $d = 5$. (C) Selection of trusted candidates using maximum likelihood. (D) Illustration of priority impute via window similarity (left) and allele frequency (right).

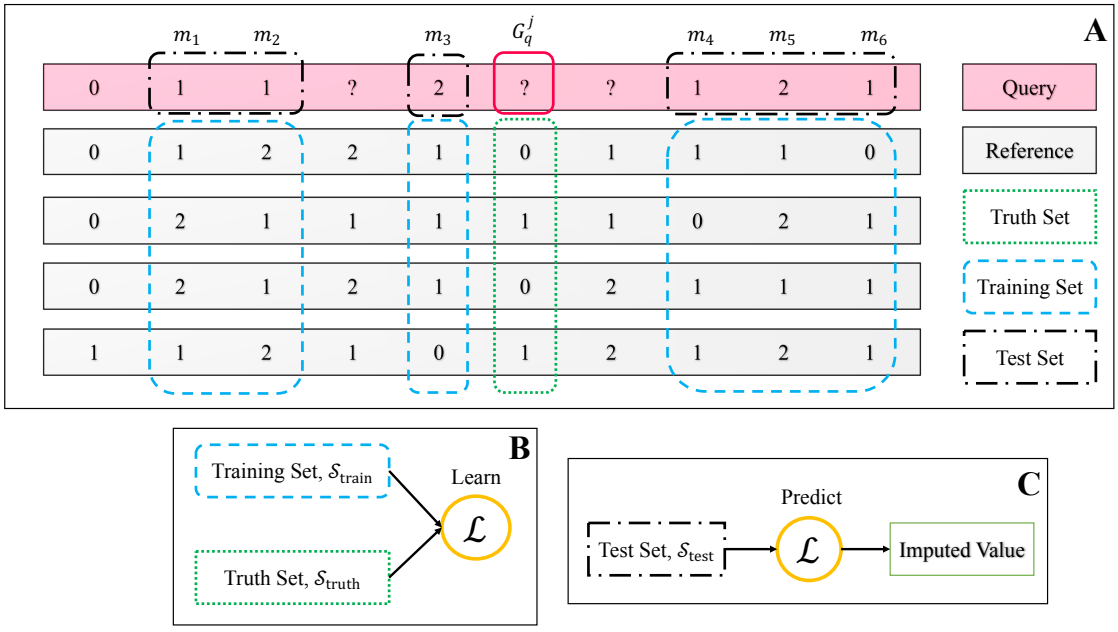


Figure 4.2: ADDIT-M for model organisms: **A.** Construction of truth (green dotted rectangle), training (blue dashed rectangle) and testing (black dashed-dot rectangle) sets from query sample (top pink block) and reference panel (gray blocks) for model organisms, assuming $d = 7$. **B.** Training procedure for supervised learning algorithm $\mathcal{L}$. **C.** Imputation procedure using trained classifier $\mathcal{L}$.

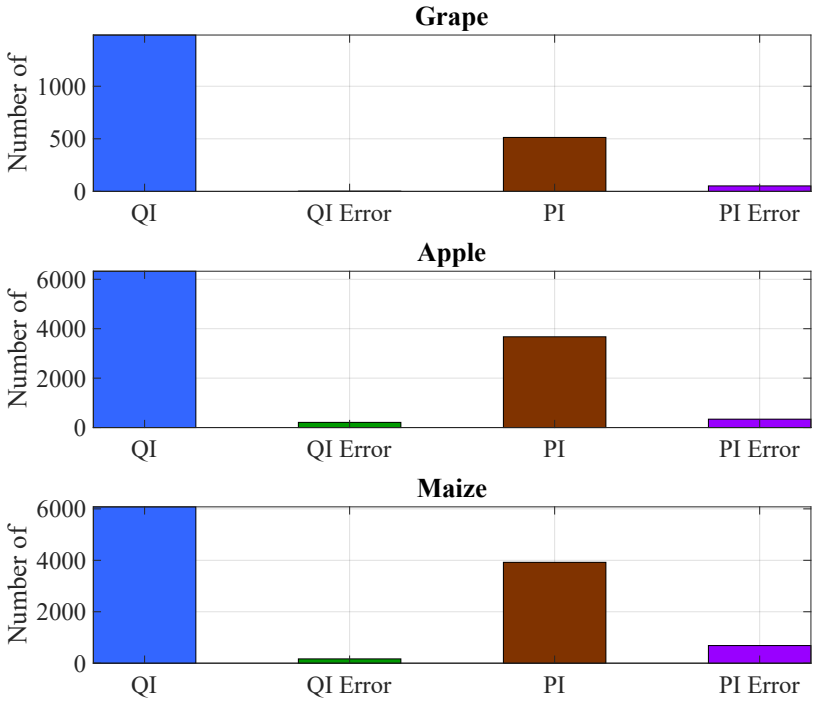


Figure 4.3: Illustration of the number of quick imputations (QI) in ADDIT-NM: blue, quick imputation error (QI Error): green, priority-based imputations (PI): dark red, and priority-based imputation error (PI Error): purple, for the non-model organisms: grape, apple, and maize.

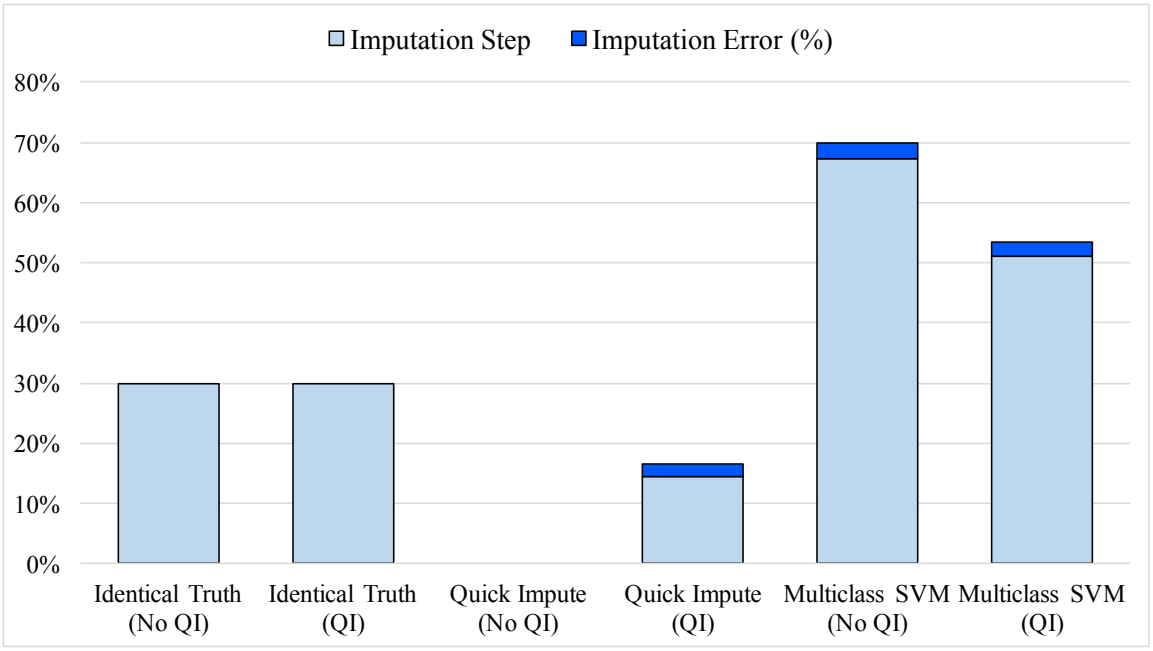


Figure 4.4: Distribution of algorithm steps (identical truths, quick impute, and multiclass classification via SVM) used for imputation of ADDIT-M with and without the quick impute step in human data. The light blue lower blocks denote the percent of missing data that are imputed via each step in the ADDIT-M formalism for model organisms. The upper dark blue blocks denote the error (%) corresponding to each of those steps.

CHAPTER 5

HYBRID CORRECTION OF ERRONEOUS LONG READS USING ITERATIVE LEARNING

The following manuscript describes the work in this chapter.

- O. Choudhury, A. Chakrabarty, S. Emrich. *A Hybrid Error Correction Algorithm for Long Reads with Iterative Learning.* Submitted, Bioinformatics, 2017

5.1 Background

Second-generation sequencing technologies, namely Illumina [15] and 454 pyrosequencing [105], have fueled the use of sequencing-driven scientific research by inexpensively generating highly accurate 'reads' or DNA sequence fragments. The trade-off is that the reads are relatively shorter, and thus not ideal for more difficult *de novo* tasks such as complex genome assembly and the reconstruction of full length mRNA isoforms [14].

Third-generation sequencing techniques introduced by Pacific Biosciences [45, 78] and more recently Oxford Nanopore [27, 46, 102] generate longer reads – thousands of bp on average and as high as 23,000 bp [76]. Further, these sequencing methods are not subject to the amplification and compositional biases often experienced with second-generation sequencing [28, 122]. Long reads also can overcome challenges associated with some repetitive regions and large transcript isoforms if they are longer than the problematic region(s). Their primary trade-off is the generated long reads are much less accurate with estimated error rates as high as 20% with PacBio [134, 136] and 35% using Oxford Nanopore [52].

A number of correction methods have been developed to address these high error rates. The self-correction tool HGAP [29], for example, computes multiple alignments of high coverage long reads to determine corrections. Alternative hybrid correction tools like LSC [14], PacBioToCA [76], LoRDEC [121], proovread [56], and CoLoRMap [57] utilize accurate, high quality short reads that were generated from the same or related samples to identify and make corrections. Nanocorr [52], the hybrid error correction tool designed to correct Oxford Nanopore reads, uses high quality Illumina MiSeq reads.

The currently available methods, however, may not generate an optimal solution for every erroneous base, especially when more localized information (quality and possible variant information between individuals) from the alignments is available. For example, the authors in [73] have emphasized the importance of incorporating quality values while correcting noisy sequence data. Although proovread involves an iterative mapping step where disjoint subsets of the short reads are mapped to improve sensitivity, an iterative approach where low-confidence corrections are further investigated has not been studied before.

Here, we present HECIL, a new hybrid error correction algorithm that determines an optimal correction policy over a convex combination of decision weights based on base quality and mapping identity of the aligned short reads. It also employs a novel iterative learning approach that leverages updated context from prior iterations in current iterative correction. We test HECIL on real prokaryotic and eukaryotic datasets: the bacteria *Escherichia coli*, the fungus *Saccharomyces cerevisiae*, and one of the most important malaria vectors *Anopheles funestus*. We compare its performance with the state-of-the-art hybrid error correction tools proovread, LoRDEC, and CoLoRMap. For an overwhelming majority of evaluation metrics HECIL outperforms the competing tools. The optional extension of iterative learning-based correction further improves its accuracy by a significant margin.

The contributions of this work include:

- development of a hybrid error correction algorithm that incorporates underlying information (base quality and mapping identity) of short reads;
- assessment of reliability of aligned short reads before leveraging them for error correction;
- design of an iterative learning approach that introduces a confidence metric to each correction for further investigation;
- better performance than state-of-the-art error corrections tools when tested on real, heterozygous, low-coverage data sets, including the important malaria vector *Anopheles funestus*;

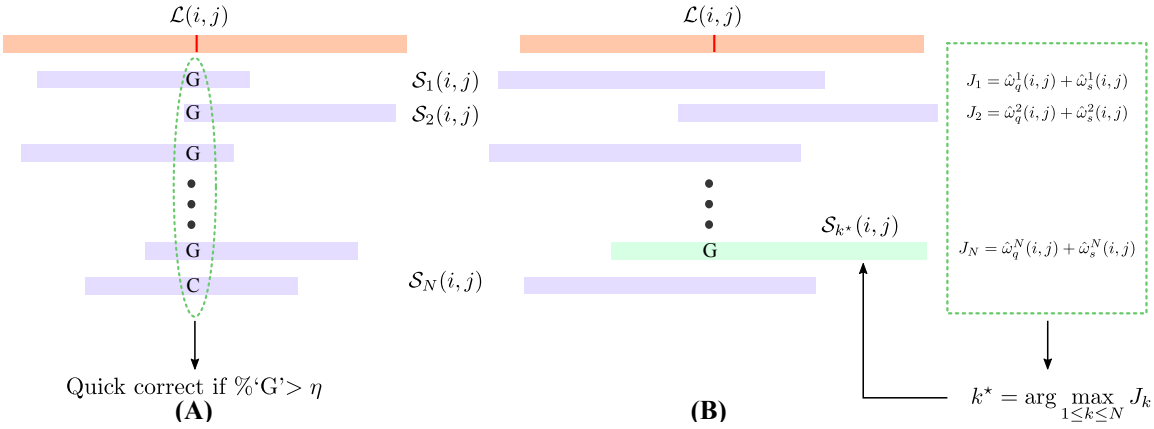


Figure 5.1: Illustration of Steps 1 and 2 of HECIL's core algorithm. The orange rectangle denotes an erroneous long read and the purple rectangles represent aligned short reads. (**A**) Illustration of Quick Correction with high consensus (**B**) Illustration of Optimization-based Correction. The green dashed box depicts the objective function values, from which the optimal short read (green rectangle) is selected for correction.

5.2 Methods

5.2.1 Overview

Our proposed algorithm, HECIL, works under the assumption that we can access long and short reads from the same colony. That is, we assume that all reads (long and short) are derived from highly similar individuals: this ensures that the short reads are reliable sources of information to inform the correction procedure. This assumption is standard amongst all hybrid error correction algorithms, see for example [14, 57]. We begin by aligning the given set of short reads to the long reads. For each alignment, we compute normalized weights using base quality information and alignment identity of the underlying short reads. The short read that maximizes the sum of weights is used for correction; in this manner, we tend to select higher quality short reads that have a suitable degree of overlap with the corresponding region in the long read.

We bolster our correction framework by formulating an iterative learning paradigm to achieve higher accuracy. In the iterative learning framework, the core algorithm discussed above is run iteratively, with each iteration learning from the data sifted through in the prior iteration. Specifically, at each iteration, we assign each correction with a corresponding confidence value. This enables us to flag low-confidence corrections, which can be re-corrected in subsequent iterations by leveraging additional information obtained from high-confidence corrected subregions of the long reads. Although multiple iterations increase total computation time, they cause a significant improvement in accuracy.

5.2.2 The Core Algorithm

5.2.2.1 Step 1: Quick Correction

We begin by performing alignment in order to detect the location of errors on the long reads and determine the set of putative corrections. As suggested in [56,

57], we obtain these alignments using BWA-MEM [87]. Following alignment, we mark the positions of disagreement (mismatches, insertions, and deletions) on long reads as questionable or erroneous. For each erroneous position on the long read, we investigate the set of short reads that align to it. If an erroneous base aligns to a single short read, or there is a strong consensus (determined by a threshold $\eta \in (0, 1]$ selected by the user), we replace the base on the long read with the respective aligned base of the short read. This is illustrated in Figure 5.1A. By performing this step, we quickly correct erroneous bases with high confidence, which leads to a reduction in the search space for the next step. Note that this step is inspired by the effectiveness of the majority voting method proposed in [14]. However, unlike majority voting, which requires a consensus of $> 50\%$, we adopt a stricter threshold of $\eta > 0.9$, and perform quick correction only if the consensus is above this high threshold. We will explain why such a high value of η is required in the following subsection.

5.2.2.2 Step 2: Optimization-based Correction

For the remaining identified erroneous bases, we use an optimization-based correction framework. Let $\mathcal{L}(i, j)$ be the jth erroneous base corresponding to the ith long read. Suppose N short reads align to the erroneous base $\mathcal{L}(i, j)$, and $\{\mathcal{S}_k(i, j)\}_{k=1}^{N}$ denote the set of aligning short reads.

For each $k = 1, 2, \ldots, N$, we assign two normalized weights $\hat{\omega}_q^k(i, j)$ and $\hat{\omega}_s^k(i, j)$. The weight $\omega_q(i, j)$ is determined by extracting the PHRED quality score which is readily available from FASTQ files. The normalized quality weight is then computed using

$$\hat{\omega}_q^k(i, j) := \frac{\omega_q^k(i, j)}{\max_{1 \leq k \leq N} \omega_q^k(i, j)}.$$

Next, we evaluate the normalized similarity weight $\hat{\omega}_s^k(i, j)$ by computing the alignment identity, defined as the number of exact matches of the short read $\mathcal{S}_k(i, j)$ to the

long read $\mathcal{L}(i,j)$, divided by the length of the short read. It is important to mention that all short reads are of equal length.

For each short read, we compute a cost function by taking a convex combination of the two weights

$$J_k(i,j) = \frac{1}{2}\left(\hat{\omega}_q^k(i,j) + \hat{\omega}_s^k(i,j)\right).$$

We then solve the following optimization problem

$$k^\star = \underset{1\leq k\leq N}{\arg\max}\ \left(\hat{\omega}_q^k(i,j) + \hat{\omega}_s^k(i,j)\right). \tag{5.1}$$

which yields the index $k^\star$ of the short read $S_{k^\star}(i,j)$ with the maximum combined quality and similarity weight[1]. Note that the optimal cost for each $\mathcal{L}(i,j)$ is denoted by $J_{k^\star}(i,j)$. Subsequently, we replace the erroneous base $\mathcal{L}(i,j)$ on the long read with the corresponding base of the short read $\mathcal{S}_{k^\star}(i,j)$. This is illustrated in Figure 5.1(B).

For highly heterozygous samples (e.g., mosquitoes), low frequency bases in aligned short reads may indicate inherent variation, not necessarily sequencing errors. Correction algorithms that solely rely on a consensus call or majority vote often discard these heterogenous alleles. The optimization in Step 2 of HECIL is not biased by bases which have high frequency, and hence, is better able to capture variation between similar individuals. This is corroborated by the results obtained from testing HECIL on the highly heterozygous mosquito data set of *Anopheles funestus* in Section 5.3.

At this junction, we can explain the rationale behind choosing a high consensus threshold for Step 1. If one achieves perfect consensus ($\eta = 1.0$) of the short reads, then all short reads recommend the same base for correction. In such a scenario, implementing Step 2 is redundant, because it will arrive at the same conclusion after

[1]In case there is a conflict amongst maximizers, the short read with highest quality is selected to be the winner.

computing weights over multiple short reads. Thus, Step 1 can be used to preempt computations that would otherwise be incurred in Step 2. Relaxing the value of η to within $(0, 1)$ results in higher number of quick corrections, but the probability of agreeing with the outcome of Step 2 is $1 - \eta$, implying that η should be chosen close to 1. We must mention that this does not imply that the consensus value is the correct base for correction purposes, merely that, *given the short read data*, we can avoid unnecessary computations with high probability using this high-threshold majority vote step.

Another important point to note is that the weighting factors, quality and similarity, are not contending objectives; that is, a high quality read does not necessitate high similarity. Thus, we have to consider a combination of these weights as in equation (5.1) rather than formulating the problem in a multi-objective optimization framework and searching for Pareto-optimal solutions.

5.2.3 Improvement Of Correction Via Iterative Learning

We employ the terminology *iterative learning* from the control theory literature, namely, from the principle of iterative learning control proposed in [72]; see [7] for an excellent survey. The definition of iterative learning that best explains our motivation in this work is found in [12]: iterative learning *considers systems that repetitively perform the same task with a view to sequentially improving accuracy.* The *same task* in our proposed framework refers to the core algorithm of HECIL.

Although this formalism bears a resemblance to the "on-line learning" framework in the machine learning literature [126], where data for improving the hypothesis becomes available iteratively, there is a key difference. The difference is that in iterative learning, the data used to improve the hypothesis is a product of the outcomes of the prior hypothesis and a metric assigned to the quality of these outcomes. Conversely, in on-line learning, the sequentially available data stream is usually generated by an

exogenous system.

Herein, we explain how we use iterative learning to improve error corrections at the ℓth iteration by learning from high-confidence corrections in the $(\ell-1)$th iteration. The overall iterative learning scheme is illustrated in Figure 5.2.

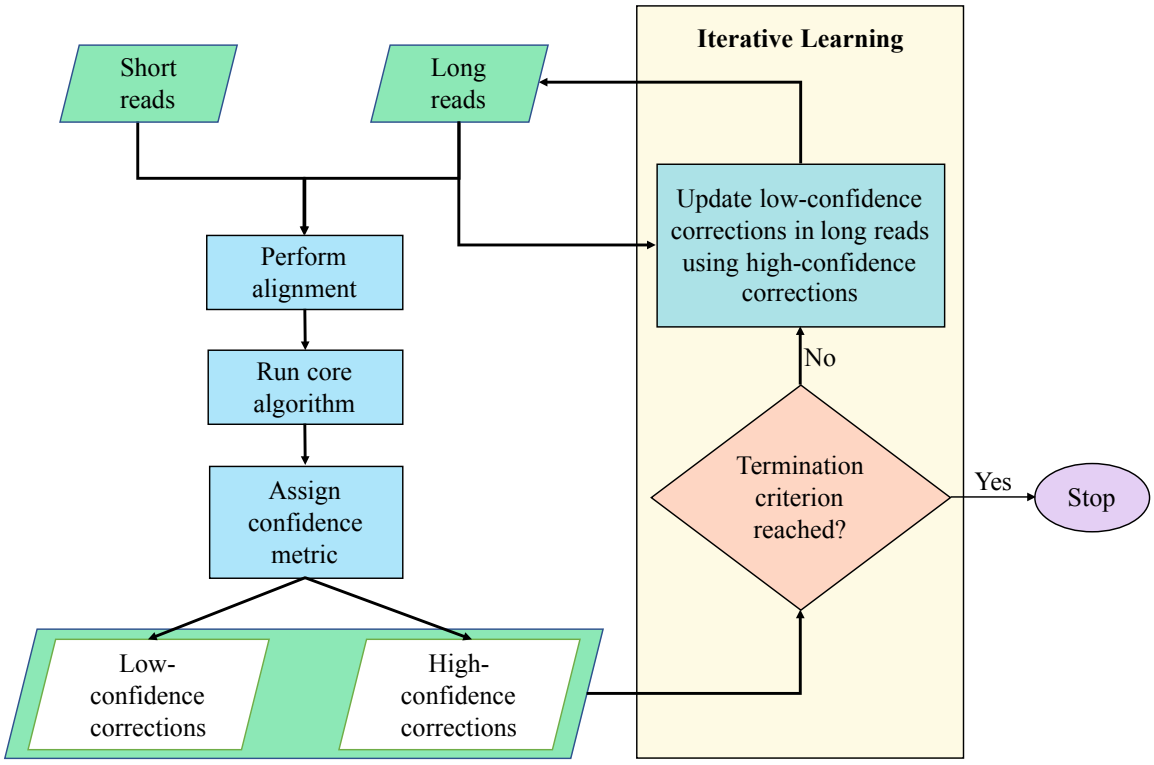


Figure 5.2: Illustration of iterative learning procedure with the HECIL core algorithm as the error correction method.

5.2.3.1 Assignment Of Confidence

For each $\mathcal{L}(i,j)$ in the ℓth iteration, suppose the corresponding optimal cost obtained by solving (5.1) be denoted by $J^{(\ell)}_{k^\star}(i,j)$, and let $\mu^{(\ell)}$ denote the α-percentile computed over all these optimal costs. We usually select $\alpha > 0.8$ so that a small per-

centage of the optimal corrections are considered to be of high confidence. Choosing α small could increase the likelihood of spurious corrections tainting the realignment stage, described next.

5.2.3.2 Realignment Based On High-Confidence Corrections

This is the most crucial stage for embedding information from high-confidence corrections into subsequent iterations. At each iteration, we compute confidence values as discussed above. We correct the long read at the sites $\mathcal{L}(i, j)$ where the corresponding optimal short read $S_{k^\star}(i, j)$ exhibits high confidence. We *learn* from this embedded information in successive iterations by re-aligning the updated long reads to the short reads. As we use alignment information to determine the optimal correction policy for long reads, after each iterative correction, the updated long reads exhibit higher consensus or similarity with the short reads.

Note that, for each iteration, the updated context of $\mathcal{L}(i, j)$ may generate different sets of aligned short reads, leading to different sets of normalized weights $\omega^k_{q_{ij}}$ and $\omega^k_{q_{ij}}$. This is why the confidence threshold $\mu^{(\ell)}$ is recomputed based on the statistics of the optimal costs, not fixed for all iterations. The sites on the long read corresponding to low-confidence short reads are left uncorrected in the current iteration. In the next iteration, these low-confidence sites are corrected via the core algorithm.

5.2.3.3 Termination Criteria

We propose two criteria for termination. The first criterion is based on the number of unique k-mers calculated after each iterative correction. If the difference in the number of k-mers between two successive iterations is below a given threshold ε, then we terminate. A particular advantage of selecting this evaluation metric for determining the termination condition is that it offers a robust, reference-free approach [25].

The second criterion involves pre-calculating the number of iterations $n \in \mathbb{N}$ required to generate at least M' high-confidence corrections from M erroneous positions. We consider corrections to be of high-confidence if they are in the top α-percentile of the optimal short reads. By definition of percentiles, this implies that $(1-\alpha) \times 100\%$ of the short reads are retained as high-confidence corrections, while $\alpha \times 100\%$ of the short reads are discarded. Therefore, after the first iteration, αM low-confidence short reads remain for the next iteration of learning and error correction. In general, the number of low-confidence short reads remaining after n iterations will be $\alpha^n M$. If the user requires at least M' high-confidence corrections to be made iteratively, one needs to select n that satisfies $M' < (1-\alpha^n)M$. In such a case, algebraic manipulations yield the following bound on the required number of iterations

$$n > \frac{1}{\log \alpha} \log \left| 1 - \frac{M'}{M} \right|. \tag{5.2}$$

Note that both logarithm terms are negative.

If either of these two criteria is triggered, we terminate the iterative learning procedure.

5.3 Results And Discussion

5.3.1 Data Acquisition

We test the performance of HECIL on three datasets of varying size: the bacterial genome of *E. coli*, the fungal genome of *S. cerevisiae*, and the malarial vector genome of *A. funestus*. We explore PacBio-sequenced long reads, Illumina-sequenced short reads, and reference genomes of *E. coli* and *S. cerevisiae*, as suggested in the supple-

mentary material of [57]. Long reads[2] of *E. coli* are filtered to exclude reads shorter than 100 bp. The final set contains 33,360 reads that total 98 million bases (Mbp). The corresponding short reads (accession ID: ERR022075) comprise 22,720,100 reads that are 202 bp long. The strain K-12 substr. MG1655 is used for our alignment-based validation of HECIL. To test *S. cerevisiae* data we use 1,758,169 long reads, with 1,402 Mbp[3] and 4,503,422 short reads (accession ID: SRR567755). The reference genome of strain S288C is 12.2 Mbp in size. We obtain long reads for *A. funestus*[4], comprising data from 44 flowcells, ranging between 59,937 and 244,754 reads. Due to the high computational time required by proovread and CoLoRMap to correct the reads of all flowcells, we present a comparative analysis based on a random selection of three: flowcells 1, 4, and 16. Short sequences (accession ID: SRR630594) consists of 37,797,235 reads. The reference genome of strain Fumoz (GenBank assembly accession: GCA_000349085.1) is used for validating the correction algorithms.

5.3.2 Computational Set-up

The experiments were run on Dell PowerEdge R815 servers with AMD Opteron processor 6378, Quad 16 core 2.4 GHz CPU, 32 cores, 512 GB RAM, and 2 x 300 GB 15K RPM SAS drives. We use the Unix *time* command to record the runtime and maximum memory footprint of each tool. To compare our proposed algorithm against the state-of-the-art, we compute fine-grained (k-mer) and coarse-grained evaluation metrics for each dataset. For the coarse-grained metrics, we consider alignment-based and assembly-based results. For mapping the long reads to its corresponding reference genome, we use BLASR [26] due to its capability in detecting longer alignments.

[2] https://github.com/PacificBiosciences/DevNet/wiki/E-coli-K12-MG1655-Hybrid-Assembly

[3] https://github.com/PacificBiosciences/DevNet/wiki/Saccharomyces-cerevisiae-W303-Assembly-Contigs

[4] https://gembox.cbcb.umd.edu/seqdata/afun_new_raw.tgz

5.3.3 Evaluation Metrics

5.3.3.1 k-mer-based

We use the popular k-mer counting tool Jellyfish [103] to compute the number of unique k-mers, obtained after each error correction algorithm. For a given DNA sequence, k-mers are overlapping substrings of length k. Unlike short reads, errors in long reads are uniformly distributed across their length. As the probability of a random error occurring multiple times is low, long reads result in large number of unique k-mers. The authors in [25] reported that the set of common k-mers between the highly accurate short reads and the erroneous long reads were crucial in improving the quality of data for downstream analysis. Therefore, a correction algorithm that reduces the number of k-mers and unique k-mers while increasing the number of valid k-mers is desirable. Figure 5.3 gives an illustrative example of this idea using the results of *A. funestus*, flowcell #16.

5.3.3.2 Alignment-based

After each method of correction, we align corrected long reads to its reference genome using BLASR [26]. In addition to computing the number of aligned reads and aligned bases, we evaluate percentage of matched bases by the ratio of total number of matched bases and length of sequences in the long reads. We calculate percent identity (PI) by the ratio of matches to alignment length. We also determine the number of aligned reads with percent identity above a given threshold (say, 90%).

5.3.3.3 Assembly-based

One of the most important downstream applications of long reads is *de novo* genome assembly. For this purpose, we use the assembler Canu [77], specifically designed for noisy long reads. We then use QUAST [55] to evaluate the quality of

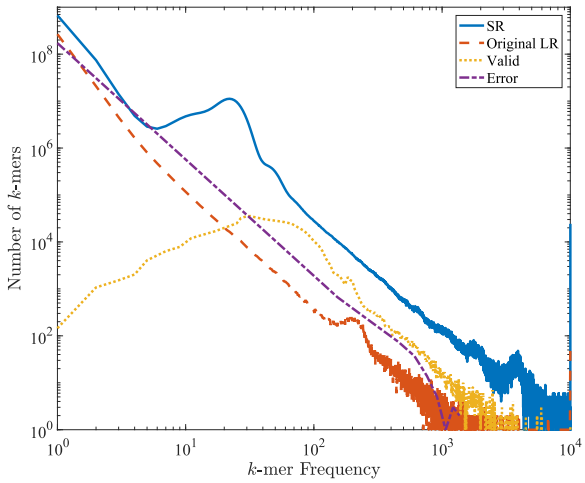


Figure 5.3: Distribution of k-mer frequency (k=17) in *Anopheles funestus*, flowcell #16. The x and y-axes denote k-mer frequency and count of frequency, respectively. The continued blue line and dashed red line represent k-mers generated from short reads (SR) and original long reads (Original LR), respectively. As discussed in Section 5.3.3.1, the dotted yellow line indicates an increase in valid or corrected k-mers along with reduced low frequency k-mers. The error k-mers, shown in purple dot-dashes, mostly comprise low frequency k-mers.

genome assembly. We measure the total number of contigs, length of the longest contig, and total length, i.e., total number of bases in the assembly. We also report the values of N50 (length for which the collection of all contigs of that length or longer covers at least half an assembly), and NG50 (length for which the collection of all contigs of that length or longer covers at least half the reference genome).

5.3.4 Comparative Analysis

We compare the performance of HECIL with existing hybrid error correction tools such as proovread-2.14.0, LoRDEC-0.6, and CoLoRMap. We use the above-mentioned k-mer-based, alignment-based, and assembly-based metrics to assess the efficacy of each approach. The comparative results for k-mer-based and alignment-based parameters are presented in Table 5.1. We report the parameters before correction (original) and after each method of error correction. As discussed earlier, correction of long reads would reduce the total number of k-mers and unique number

of k-mers. Long reads corrected by HECIL generate the lowest number of k-mers, with the exception of the data set *A. funestus* - flowcell 4, where it is still comparable to the best results obtained from proovread. An increase in valid k-mers indicates higher consensus to the accurate short reads, hence higher accuracy of the corrected long reads. For all data sets, HECIL consistently produces more valid k-mers. To evaluate the alignment-based metrics, we align the original and corrected long reads of each data set to its corresponding reference genome. HECIL results in highest number of aligned bases and reads, leading to highest quality of alignment.

We present the results of assembly-based metrics in Table 5.2. For *E. coli* and yeast, HECIL generates more contiguous assembled long reads, compared to the existing tools, except CoLoRMap. The size of the longest contig and the number of bases in the assembled data are highest for our proposed approach. Furthermore, the standard assembly quality parameters like N50, and NG50 have highest values for HECIL. As discussed in [57], CoLoRMap performs better than proovread and LoRDEC, when tested on *E. coli* and *S. cerevisiae*. HECIL outperforms the other correction tools, with the exception of flowcell #4 of mosquito data set, where it still exhibits comparable performance with respect to proovread. However, for alignment-based metrics, it performs better than proovread in generating higher quality *de novo* assembly.

Although performance of hybrid correction algorithms largely depends on the set of high coverage short reads, we devise additional experiments to show that this is not a significant constraint for HECIL. We downsample short reads by randomly selecting 50%, 25%, and 12% of the data to be used for correction. In *E. coli*, this results in a subset of short reads for correction with an average coverage of 62x, 33x, and 18x, respectively. In Table 5.3, we present k-mer-based and alignment-based parameters from correcting long reads of *E. coli* with the downsampled short reads using HECIL. We observe that although higher coverage of underlying short reads

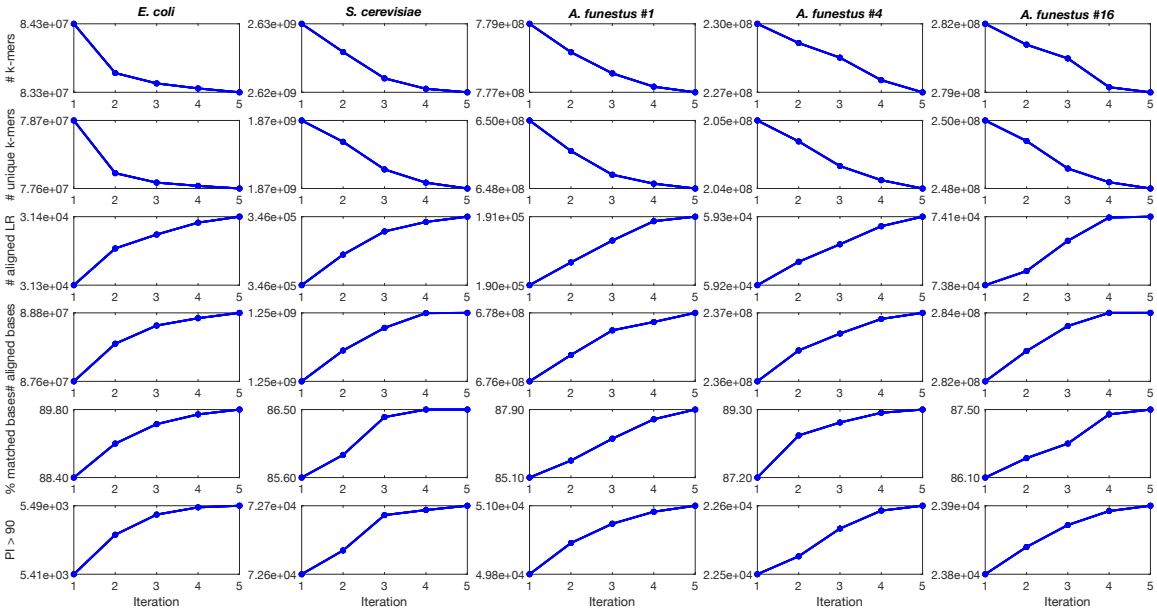


Figure 5.4: Improvement of evaluation metrics for different data sets with iterative learning (up to 5 iterations). Note that number of k-mers and number of unique k-mers are monotonically decreasing with increasing number of iterations, whereas the other metrics consistently show monotonic increment.

generates better results, we still achieve comparable results while using lower coverage reads. We further determine assembly-based parameters from the lowest coverage (18x or 12%) short reads, as shown in Table 5.2. It is evident that HECIL still outperforms the other tools and is least affected by the diminished coverage of short reads. Hence, HECIL can be potentially used to attain good results in projects that do not have high coverage short read data readily available (e.g., larger eukaryotic genomes processed mostly with long reads).

In Table 5.4, we compare the runtimes and maximum memory usage incurred in correcting each data set using proovread, LoRDEC, CoLoRMap, and HECIL. Runimes and maximum memory were recorded using the *time* command in Unix. proovread, LoRDEC, and CoLoRMap were run with 16 threads. The workload of HECIL was split into 20 concurrent tasks, that were run in parallel. Computation time of hybrid error correction methods is mainly dominated by the underlying

steps of generating intermediate data. For instance, a prerequisite of proovread, CoLoRMap, and HECIL is the mapping of short reads to the long reads, that accounts for a significant fraction of the overall runtime. Similarly, LoRDEC and CoLoRMap construct a graph data structure, which demands high computational resources. However, LoRDEC uses the efficient GATB library [3], which lowers the overhead, as shown in Table 5.4. Quick correction, the first step of HECIL, prunes the search space before running the core algorithm. Although our tool incurs higher computation time than LoRDEC, it is significantly faster than the other correction methods and generates overall higher quality corrected long reads.

5.3.5 Effect Of Iterative Learning

As the runtime of HECIL is comparable to the fastest tool (Table 5.4), we leverage our proposed iterative learning scheme to further improve its correction accuracy. We select a high-confidence cut-off of 85 percentile, that is, $\alpha = 0.85$. The results obtained after each iterative correction of HECIL is presented in Figure 5.4. For each data set (each column), we observe that the evaluation metrics: number of k-mers, number of unique k-mers, number of aligned long reads, number of aligned bases, percent of matched bases, and number of long reads with percent identity higher than 90%, improve after each iteration, until one of the termination criteria is reached. For the termination criteria, we select $|\varepsilon|$ as 0.02 and choose $M'/M = 0.5$, which, using (5.2) results in at least $n = 5$ iterations. Hence, the iterative learning-based extension of HECIL further improves corrected read quality.

5.4 Conclusion

Third-generation sequencing techniques, particularly Single-Molecule Real-Time (SMRT) sequencing, is revolutionizing modern biology by generating considerably

longer DNA sequences with lower amplification errors when compared to its second-generation counterparts. The usefulness of current long read data, however, is restricted due to karge amount of sequencing error. Thus, it is crucial to apply correction methods prior to applications like *de novo* genome assembly.

Here we propose a novel approach of hybrid error correction, called HECIL, which corrects erroneous long reads based on optimal combination of base quality and mapping identity of aligned short reads. We further propose an iterative learning framework that leverages information from prior iterations in successive iterative corrections. When tested on real data sets of *E. coli*, yeast, and the malarial mosquito *Anopheles funestus*, HECIL exhibits better performance than the state-of-the-art error correction tools proovread, LoRDEC, and CoLoRMap, for an overwhelming majority of evaluation metrics. The low computation time of HECIL enabled us to achieve superior performance within acceptable time. To the best of our knowledge, an iterative strategy for improving correction quality via informed realignment is proposed in this work for the first time. The method shows potential using the HECIL core algorithm, but can also be seamlessly integrated with other error correction algorithms.

TABLE 5.1:

COMPARISON OF ALIGNMENT BASED METRICS

Data	Evaluation Metric	Original	proovread	LoRDEC	CoLoRMap	HECIL (Iteration 1)
E. coli	# k-mers	87,200,204	84,416,389	86,425,411	86,588,771	**84,294,561**
	# unique k-mers	81,523,648	78,925,288	80,708,419	80,399,425	**78,693,704**
	# valid k-mers	14,531,881	11,463,127	10,240,970	15,026,950	**15,973,826**
	# aligned reads	31,071	23,453	30,837	31,271	**31,332**
	# aligned bases	86,642,500	71,320,858	79,365,407	83,344,272	**87,582,014**
	% matched bases	76.9	87.9	85.2	87.5	**88.4**
	# reads with PI > 90%	4,190	1,742	4,832	4,430	**5,408**
S. cerevisiae	# k-mers	2,630,997,642	2,627,421,712	2,626,738,639	2,628,138,521	**2,626,390,081**
	# unique k-mers	1,870,396,869	1,871,451,237	1,868,238,946	1,869,232,456	**1,867,828,519**
	# valid k-mers	36,904,129	32,436,294	30,534,546	37,797,300	**39,452,743**
	# aligned reads	224,694	222,976	221,692	223,641	**346,242**
	# aligned bases	1,229,724,663	1,205,706,114	1,171,490,123	1,207,729,568	**1,247,616,674**
	% matched bases	78.8	83.1	83.4	**85.6**	**85.6**
	# reads with PI > 90%	70,346	27,342	70,529	42,136	**72,562**
A. funestus (Flowcell # 1)	# k-mers	809,767,884	778,931,843	782,845,319	783,270,385	**778,574,946**
	# unique k-mers	692,831,731	649,989,172	653,931,808	662,366,838	**649,764,906**
	# valid k-mers	211,908,809	172,074,427	229,625,736	222,195,325	**242,957,349**
	# aligned long reads	190,217	94,536	190,240	190,166	**190,229**
	# aligned bases	671,881,278	401,850,047	655,072,426	660,848,583	**676,055,060**
	% matched bases	84.0	81.4	83.1	82.1	**85.1**
	# reads with PI > 90%	30,487	4,096	36,364	13,220	**49,810**
A. funestus (Flowcell # 4)	# k-mers	240,722,003	**229,582,178**	231,586,667	233,953,119	229,611,352
	# unique k-mers	216,327,700	**205,053,236**	205,883,182	206,986,374	205,064,188
	# valid k-mers	80,612,612	72,716,589	82,568,831	81,027,437	**83,788,157**
	# aligned long reads	59,163	32,726	59,165	59,159	**59,177**
	# aligned bases	231,326,514	149,049,154	234,098,182	233,435,402	**235,620,667**
	% matched bases	86.3	83.2	87.0	85.6	**87.2**
	# reads with PI > 90%	20,253	2,094	22,042	12,526	**22,483**
A. funestus (Flowcell # 16)	# k-mers	295,751,035	282,128,190	283,971,486	286,819,483	**281,523,755**
	# unique k-mers	265,998,542	250,267,133	252,291,701	254,293,778	**249,528,780**
	# valid k-mers	96,317,177	86,396,798	106,713,483	101,431,900	**109,954,860**
	# aligned long reads	73,779	43,530	73,757	73,750	**73,790**
	No. of aligned bases	278,976,792	190,054,632	280,699,552	280,831,201	**282,244,589**
	% matched bases	84.3	82.7	85.6	84.5	**86.1**
	# reads with PI > 90%	22,111	2,164	22,945	10,418	**23,771**

TABLE 5.2:

COMPARISON OF ASSEMBLY BASED PARAMETERS

Data	Evaluation Metric	Original	proovread	LoRDEC	CoLoRMap	HECIL (Iteration 1)
E. coli	# contigs	182	26	24	**19**	**19**
	Largest contig	69,266	605,792	920,903	1,089,140	**1,223,474**
	Total length	3,508,197	4,629,719	4,623,137	4,624,793	**4,838,971**
	N50	24,663	231,774	226,456	239,066	**256,830**
	NG50	17,847	231,774	226,456	239,066	**294,635**
E. coli (Downsampled)	# Contigs	182	29	28	24	**20**
	Largest contig	69,266	567,484	885,819	813,262	**1,204,631**
	Total length	3,508,197	4,235,031	4,068,085	4,036,161	**4,596,013**
	N50	24,663	189,712	179,638	184,367	**232,826**
	NG50	17,847	212,621	190,621	210,913	**267,311**
S. cerevisiae	# contigs	26	32	28	**24**	**24**
	Largest contig	1,543,990	1,537,979	1,552,711	1,555,857	**1,558,190**
	Total length	12,341,981	12,485,995	**12,497,078**	12,315,869	12,435,702
	N50	777,602	777,713	818,962	932,935	**1,018,591**
	NG50	777,602	777,713	818,962	932,935	**1,538,190**
A. funestus (Flowcell # 1)	# contigs	256	64	65	107	**62**
	Largest contig	33,452	17,794	35,499	38,298	**100,425**
	Total length	3,149,449	135,454	764,805	831,010	**5,229,535**
	N50	13,989	12,703	14,033	9,265	**14,403**
	NG50	33,452	17,794	35,499	38,298	**100,425**
A. funestus (Flowcell # 4)	# contigs	61	60	60	**59**	**59**
	Largest contig	61,535	22,554	56,919	40,286	**70,241**
	Total length	852,550	100,624	453,058	581,736	**900,293**
	N50	16,571	15,731	15,783	16,362	**16,618**
	NG50	61,535	22,554	56,919	40,286	**70,241**
A. funestus (Flowcell # 16)	# contigs	86	63	68	77	**59**
	Largest contig	34,761	19,301	37,300	34,147	**37,851**
	Total length	506,571	101,672	514,942	503,179	**557,165**
	N50	11,367	12,445	12,275	11,797	**12,633**
	NG50	34,761	19,301	37,300	34,147	**37,851**

TABLE 5.3

COMPARISON OF METRICS WITH DOWNSAMPLING

Data	**Evaluation Metric**	**All SRs**	**50% SRs**	**25% SRs**	**12% SRs**
E. coli	# k-mers	84,294,561	84,121,231	83,986,262	83,909,706
	# unique k-mers	78,693,704	78,292,463	78,097,941	78,008,319
	# valid k-mers	15,973,826	15,889,155	15,737,641	15,576,317
	# aligned reads	31,332	31,328	31,322	31,318
	# aligned bases	87,582,014	87,359,227	87,288,475	87,196,236
	% matched bases	88.4	88.4	88.3	88.3
	PI	99.7	99.7	99.7	99.6
	# reads with PI $>$ 90%	5,408	5,358	5,298	5,223

TABLE 5.4

COMPARISON OF RUNTIME AND MAXIMUM MEMORY FOOTPRINT

Data	**Method**	**Runtime** (hh:mm:ss)	**Memory** (GB)
E. coli	proovread	6:15:37	11.4
	LoRDEC	38:53	6.2
	CoLoRMap	2:48:23	28.9
	HECIL	58:37	8.4
S. cerevisiae	proovread	20:54:15	14.5
	LoRDEC	3:43:12	6.1
	CoLoRMap	7:57:49	38.2
	HECIL	4:34:07	9.6
A. funestus (Flowcell # 1)	proovread	76:13:47	8.8
	LoRDEC	35:08:13	3.1
	CoLoRMap	90:50:12	23.4
	HECIL	41:16:28	7.4
A. funestus (Flowcell # 4)	proovread	36:32:25	7.3
	LoRDEC	11:25:05	6.7
	CoLoRMap	32:18:30	20.7
	HECIL	15:04:39	6.2
A. funestus (Flowcell # 16)	proovread	20:07:43	7.1
	LoRDEC	12:54:57	3.0
	CoLoRMap	34:21:50	23.6
	HECIL	15:49:33	6.1

CHAPTER 6

SUMMARY AND FUTURE WORK

6.1 Summary

Significant reduction in the cost of genome sequencing has led to an exponential growth of genome data, typically on the order of Giga basepairs (Gbp) per machine day. The pace of analyzing these ever-increasing data is limited by the rate of computing throughput. On the other hand, fidelity of such data is often degraded due to high volumes of missing and erroneous values. To mitigate the first challenge, we develop efficient frameworks that expedite analysis of big genome data. In this context, we also design hybrid predictive models to reduce cluster and cloud-based resource utilization when processing such data. To address the second challenge, we formulate novel data-driven algorithms to impute missing data and correct erroneous values in genomic applications.

Cluster, cloud, and grid computing have been extensively used to accelerate analysis of large-scale data. In Chapter 2, we show that efficient partitioning of data and merging of related workflows, in a distributed computing setup, can significantly improve the rate of data analysis. We propose three approaches of data partitioning: granularity, individual, and alignment-based partitioning to determine the optimal design of the workflow. We also observe that merging sequential but interrelated workflows can benefit from caching and further improve runtime. We propose full workflow caching, choke-eliminated, and merged partition-based methods of workflow fusion to enhance the level of concurrency during data processing. We test the

efficiency of our framework on the two most important applications in comparative genomics: genome alignment and variant detection. With optimal data partitioning and workflow fusion strategies, our framework significantly reduced the runtime of data analysis from 12 days to less than 2 hours, when tested on a data set of size 50 GB. We further demonstrate its utility in studying genomes of real data sets, particularly the ecologically and economically important northern red oak plant.

Commercial cloud services have gained much prominence in offering computational resources to manage and analyze large-scale projects in various domains of research. However, for harnessing such resources, end-users have to determine the runtime configurations, such as number of tasks, number of threads allocated to each task, data partitioning strategy, and so forth. In Chapter 3, we design a hybrid predictive model that estimates an optimal configuration to reduce runtime and cost of operation while executing large workloads on clusters and clouds. We incorporate regression-based technique to design an application-specific model that is embedded into a more generic system-level model. For a given class of applications that support both multithreading and distributed computing, this multi-tiered model infers optimal configurations to balance thread-level and task-level parallelisms. The model exhibits high predictive capability and enables optimal resource utilization for services like Amazon Elastic Compute Cloud and Microsoft Azure.

Genome wide association study (GWAS) is an approach of associating genetic variations with certain dieaseses. In the era of personalized medicine, GWAS offers a promising avenue for customized treatment of disease. It relies on the method of haplotype phasing for detecting linked genes on the chromosome. The quality of phasing is often degraded due to missing genotype data. Existing imputation techniques for missing genotypes are either sluggish or require external reference data set, not available for non-model organisms. In Chapter 4, we develop ADDIT (Accurate Data-Driven Imputation Technique): an accurate, fast, lightweight genotype impu-

tation algorithm that is applicable for both model and non-model organisms. For non-model organisms, we employ adaptive windows and trust metrics to include local and global context while determining the missing value. For model organisms, we use multi-class supervised learning algorithm to extricate information from the underlying reference panels. We test the efficacy of ADDIT relative to state-of-the-art imputation methods. When tested on real plant and human data sets, for varying sizes of data, proportions of missing genotypes, and sizes of training samples, ADDIT consistently outperforms the leading tools Beagle, IMPUTE2, and LinkImpute in terms of imputation accuracy, runtime, and memory footprint.

Third-generation sequencing techniques like PacBio and Oxford Nanopore produce long reads with reduced amplification errors. Despite the several advantages of long reads in downstream analysis, their applicability is limited due to high error rate. In Chapter 5, we present HECIL (Hybrid Error Correction using Iterative Learning), a novel algorithm for hybrid correction of erroneous long reads. It determines an error correction policy by selecting accurate aligned short reads that exhibit optimal combination of base quality and similarity to erroneous long reads. We perform a comparative study against state-of-the-art hybrid error correction algorithms on real data sets including *E. coli*, *S. cerevisiae*, and the malaria vector mosquito *A. funestus*. HECIL exhibits superior performance in an overwhelming majority of evaluation metrics. We also introduce a confidence-based iterative learning framework to further improve correction accuracy. At each iteration, it uses updated context of long reads from previous iterations to further investigate low-confidence corrections. We demonstrate that the iterative learning formulation results in consistent improvement of HECIL's correction capabilities with each iteration, and significantly outperforms competing methods.

6.2 Open Problems And Future Work

The data analysis framework presented in Chapter 2 can be extended to harness resources from public cloud for large data sets. In addition to the runtime and memory models in Chapter 3, designing an energy model would be a relevant extension. While designing the system-level predictive models, considering the nodes' behavior such as failure profile and response time would make it more robust. A natural extension of ADDIT and HECIL would be a multithreaded implementation of the tools that can also support distributed computing to achieve higher speedup. Although the multiclass classifier approach used in ADDIT-M generates accurate imputations, there still remains a scope to investigate other data-driven supervised learning approaches in Step 4 of the ADDIT-M algorithm. To determine the termination criterion for iterative learning in HECIL we propose the use of k-mer-based metrics. A more generic reference-free method of variant detection, as shown in [85], is a potential alternative approach. Finally, since a bulk of the computation time of HECIL is incurred during alignment of short and long reads, a more efficient method of generating the aligned information can reduce runtime of the iterative stage of correction.

BIBLIOGRAPHY

1. Amazon Elastic Compute Cloud. `https://aws.amazon.com/ec2/`. [Online; accessed 19-July-2014].

2. Windows Azure Cloud Platform. `http://www.windowsazure.com`. [Online; accessed 19-July-2014].

3. GATB library. `http://gatb-core.gforge.inria.fr`. [Online; accessed 17-February-2017].

4. Apache Hadoop. `http://hadoop.apache.org`. [Online; accessed 19-July-2014].

5. 1000 Genomes Project Consortium and others. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

6. A.-F. Adam-Blondon, O. Jaillon, S. Vezzulli, A. Zharkikh, M. Troggio, R. Velasco, J. Martinez-Zapater, et al. Genome sequence initiatives. *Genetics, Genomics, and Breeding of Grapes*, pages 211–234, 2011.

7. H.-S. Ahn, Y. Chen, and K. L. Moore. Iterative learning control: Brief survey and categorization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1099–1121, 2007.

8. R. Albers, E. Suijs, and P. H. de With. Triple-C: Resource-usage prediction for semi-automatic parallelization of groups of dynamic image-processing tasks. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

9. A. Albert. *Regression and the Moore-Penrose pseudoinverse*. Elsevier, 1972.

10. M. Albrecht, P. Donnelly, P. Bui, and D. Thain. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, page 1. ACM, 2012.

11. M. Aly. Survey on multiclass classification methods. *Neural Networks*, pages 1–9, 2005.

12. N. Amann, D. H. Owens, and E. Rogers. Iterative learning control for discrete-time systems with exponential rate of convergence. *IEE Proceedings-Control Theory and Applications*, 143(2):217–224, 1996.

13. G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

14. K. F. Au, J. G. Underwood, L. Lee, and W. H. Wong. Improving PacBio long read accuracy by short read alignment. *PloS one*, 7(10):e46679, 2012.

15. D. R. Bentley. Whole-genome re-sequencing. *Current opinion in genetics & development*, 16(6):545–552, 2006.

16. B. L. Browning and S. R. Browning. A fast, powerful method for detecting identity by descent. *The American Journal of Human Genetics*, 88(2):173–182, 2011.

17. B. L. Browning and S. R. Browning. Genotype imputation with millions of reference samples. *The American Journal of Human Genetics*, 98(1):116–126, 2016.

18. S. R. Browning and B. L. Browning. Haplotype phasing: existing methods and new developments. *Nature Reviews Genetics*, 12(10):703–714, 2011.

19. P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain. Work queue+ python: A framework for scalable scientific ensemble applications. In *Workshop on Python for High Performance and Scientific Computing at SC11*, 2011.

20. P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain. Work queue+ python: A framework for scalable scientific ensemble applications. In *Workshop on python for high performance and scientific computing at sc11*, 2011.

21. M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. 1994.

22. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. 1994.

23. R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

24. R. L. Cairn, M. Stoneking, and A. . Wilson. Mitochondrial dna and human evolution. *Nature*, 325(3), 1987.

25. A. B. Carvalho, E. G. Dupim, and G. Goldstein. Improved assembly of noisy long reads by k-mer validation. *Genome Research*, 26(12):1710–1720, 2016.

26. M. J. Chaisson and G. Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC bioinformatics*, 13(1):238, 2012.

27. G. M. Cherf, K. R. Lieberman, H. Rashid, C. E. Lam, K. Karplus, and M. Akeson. Automated forward and reverse ratcheting of DNA in a nanopore at 5-A precision. *Nature biotechnology*, 30(4):344–348, 2012.

28. C.-S. Chin, J. Sorenson, J. B. Harris, W. P. Robins, R. C. Charles, R. R. Jean-Charles, J. Bullard, D. R. Webster, A. Kasarskis, P. Peluso, et al. The origin of the Haitian cholera outbreak strain. *New England Journal of Medicine*, 364 (1):33–42, 2011.

29. C.-S. Chin, D. H. Alexander, P. Marks, A. A. Klammer, J. Drake, C. Heiner, A. Clum, A. Copeland, J. Huddleston, E. E. Eichler, et al. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature methods*, 10(6):563–569, 2013.

30. E. K. Chong and S. H. Zak. *An introduction to optimization*, volume 76. John Wiley & Sons, 2013.

31. O. Choudhury, N. Hazekamp, D. Thain, and S. Emrich. Accelerating Comparative Genomics Workflows in a Distributed Environment with Optimized Data Partitioning. In *C4Bio at CCGrid*.

32. O. Choudhury, N. Hazekamp, D. Thain, and S. Emrich. Accelerating comparative genomics workflows in a distributed environment with optimized data partitioning. *C4Bio for CCGrid*, 2014.

33. O. Choudhury, N. Hazekamp, D. Thain, and S. Emrich. Accelerating comparative genomics workflows in a distributed environment with optimized data partitioning and workflow fusion. *Scalable Computing: Practice and Experience*, 16(1):53–70, 2015.

34. O. Choudhury, D. Rajan, N. Hazekamp, S. Gesing, D. Thain, and S. Emrich. Balancing thread-level and task-level parallelism for data-intensive workloads on clusters and clouds. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, pages 390–393. IEEE, 2015.

35. O. Choudhury, A. Chakrabarty, and S. J. Emrich. Hapi-gen: Highly accurate phasing and imputation of genotype data. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 78–87. ACM, 2016.

36. O. Choudhury, A. Chakrabarty, and S. J. Emrich. Highly accurate and efficient data-driven methods for genotype imputation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, PP(99):1–1, 2017. ISSN 1545-5963. doi: 10.1109/TCBB.2017.2708701.

37. P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*, 38(6):1767–1771, 2010.

38. O. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, et al. The variant call format and vcftools. *Bioinformatics*, 27(15):2156–2158, 2011.

39. A. Darling, L. Carey, and W.-c. Feng. The design, implementation, and evaluation of mpiBLAST. *proceedings of ClusterWorld*, 2003:13–15, 2003.

40. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

41. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977.

42. M. DePristo, E. Banks, R. Poplin, K. Garimella, J. Maguire, C. Hartl, A. Philippakis, G. del Angel, M. Rivas, M. Hanna, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491–498, 2011.

43. R. Duan, F. Nadeem, J. Wang, Y. Zhang, R. Prodan, and T. Fahringer. A hybrid intelligent method for performance modeling and prediction of workflow activities in grids. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 339–347. IEEE Computer Society, 2009.

44. R. P. Ebstein, S. Israel, S. H. Chew, S. Zhong, and A. Knafo. Genetics of human social behavior. *Neuron*, 65(6):831–844, 2010.

45. J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, et al. Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

46. M. Eisenstein. Oxford Nanopore announcement sets sequencing sector abuzz, 2012.

47. J. Ekanayake, S. Pallickara, and G. Fox. Mapreduce for data intensive scientific analyses. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 277–284. IEEE, 2008.

48. R. Ekblom and J. Galindo. Applications of next generation sequencing in molecular ecology of non-model organisms. *Heredity*, 107(1):1–15, 2011.

49. K. A. Frazer, D. G. Ballinger, D. R. Cox, D. A. Hinds, L. L. Stuve, R. A. Gibbs, J. W. Belmont, A. Boudreau, P. Hardenbol, S. M. Leal, et al. A second generation human haplotype map of over 3.1 million SNPs. *Nature*, 449(7164): 851–861, 2007.

50. J. Freese. Genetics and the social science explanation of individual outcomes 1. *American Journal of Sociology*, 114(S1):S1–S35, 2008.

51. W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE, 2001.

52. S. Goodwin, J. Gurtowski, S. Ethe-Sayers, P. Deshpande, M. C. Schatz, and W. R. McCombie. Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome research*, 25(11):1750–1756, 2015.

53. R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 455–466. ACM, 2014.

54. R. L. Grossman. The case for cloud computing. *IT professional*, 11(2):23–27, 2009.

55. A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

56. T. Hackl, R. Hedrich, J. Schultz, and F. Förster. proovread: large-scale high-accuracy PacBio correction through iterative short read consensus. *Bioinformatics*, 30(21):3004–3011, 2014.

57. E. Haghshenas, F. Hach, S. C. Sahinalp, and C. Chauve. CoLoRMap: Correcting Long Reads by Mapping short reads. *Bioinformatics*, 32(17):i545–i551, 2016.

58. O. Harismendy, P. C. Ng, R. L. Strausberg, X. Wang, T. B. Stockwell, K. Y. Beeson, N. J. Schork, S. S. Murray, E. J. Topol, S. Levy, et al. Evaluation of next generation sequencing platforms for population targeted sequencing studies. *Genome Biol*, 10(3):R32, 2009.

59. M. Hasegawa, H. Kishino, and T.-a. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of molecular evolution*, 22 (2):160–174, 1985.

60. S. Hearne, C. Chen, E. Buckler, and S. Mitchell. Unimputed GBS derived SNPs for maize landrace accessions represented in the SeeD-maize GWAS panel, 2014. [Online; accessed 21-February-2016].

61. L. A. Hindorff, P. Sethupathy, H. A. Junkins, E. M. Ramos, J. P. Mehta, F. S. Collins, and T. A. Manolio. Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. *Proceedings of the National Academy of Sciences*, 106(23):9362–9367, 2009.

62. M. M. Holland and T. J. Parsons. Mitochondrial dna sequence analysis-validation and use for forensic casework. *Forensic science review*, 11(1):21–50, 1999.

63. B. Howie, C. Fuchsberger, M. Stephens, J. Marchini, and G. R. Abecasis. Fast and accurate genotype imputation in genome-wide association studies through pre-phasing. *Nature Genetics*, 44(8):955–959, 2012.

64. B. N. Howie, P. Donnelly, and J. Marchini. A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. *PLoS Genet*, 5(6):e1000529, 2009.

65. C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

66. S. Ibrahim, B. He, and H. Jin. Towards pay-as-you-consume cloud computing. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 370–377. IEEE, 2011.

67. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pages 59–72. ACM, 2007.

68. O. Jaillon, J.-M. Aury, B. Noel, A. Policriti, C. Clepet, A. Casagrande, N. Choisne, S. Aubourg, N. Vitulo, C. Jubin, et al. The grapevine genome sequence suggests ancestral hexaploidization in major angiosperm phyla. *Nature*, 449(7161):463–467, 2007.

69. M. Janitz. *Next-generation genome sequencing: towards personalized medicine*. John Wiley & Sons, 2011.

70. S. A. Jarvis, D. P. Spooner, H. N. L. C. Keung, J. Cao, S. Saini, and G. R. Nudd. Performance prediction and its use in parallel and distributed computing systems. *Future Generation Computer Systems*, 22(7):745–754, 2006.

71. M. A. Jobling and P. Gill. Encoded evidence: Dna in forensic analysis. *Nature Reviews Genetics*, 5(10):739–751, 2004.

72. S. Kawamura, F. Miyazaki, and S. Arimoto. Iterative learning control for robotic systems. In *Proc. of IECON*, volume 84, pages 393–398, 1984.

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

74. D. C. Koboldt, K. Chen, T. Wylie, D. E. Larson, M. D. McLellan, E. R. Mardis, G. M. Weinstock, R. K. Wilson, and L. Ding. VarScan: variant detection in massively parallel sequencing of individual and pooled samples. *Bioinformatics*, 25(17):2283–2285, 2009.

75. A. Kong, G. Masson, M. L. Frigge, A. Gylfason, P. Zusmanovich, G. Thorleifsson, P. I. Olason, A. Ingason, S. Steinberg, T. Rafnar, et al. Detection of sharing

by descent, long-range phasing and haplotype imputation. *Nature Genetics*, 40 (9):1068–1075, 2008.

76. S. Koren, M. C. Schatz, B. P. Walenz, J. Martin, J. T. Howard, G. Ganapathy, Z. Wang, D. A. Rasko, W. R. McCombie, E. D. Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature biotechnology*, 30(7):693–700, 2012.

77. S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *bioRxiv*, page 071282, 2017.

78. J. Korlach, K. P. Bjornson, B. P. Chaudhuri, R. L. Cicero, B. A. Flusberg, J. J. Gray, D. Holden, R. Saxena, J. Wegener, and S. W. Turner. Real-time DNA sequencing from single polymerase molecules. *Methods in enzymology*, 472:431–455, 2010.

79. I. Lanc, P. Bui, D. Thain, and S. Emrich. Adapting bioinformatics applications for heterogeneous systems: a case study. *Concurrency and Computation: Practice and Experience*, 2012.

80. B. Langmead and S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012.

81. B. Langmead and S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012.

82. B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg. Searching for SNPs with cloud computing. *Genome biology*, 10(11):R134, 2009.

83. B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, et al. Ultrafast and memory-

efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

84. M. Lawniczak, S. Emrich, A. Holloway, A. Regier, M. Olson, B. White, S. Redmond, L. Fulton, E. Appelbaum, J. Godfrey, et al. Widespread divergence between incipient Anopheles gambiae species revealed by whole genome sequences. *Science*, 330(6003):512–514, 2010.

85. R. M. Leggett, R. H. Ramirez-Gonzalez, W. Verweij, C. G. Kawashima, Z. Iqbal, J. D. Jones, M. Caccamo, and D. MacLean. Identifying and classifying trait linked polymorphisms in non-reference species by walking coloured de Bruijn graphs. *PloS one*, 8(3):e60058, 2013.

86. T. J. Ley, E. R. Mardis, L. Ding, B. Fulton, M. D. McLellan, K. Chen, D. Dooling, B. H. Dunford-Shore, S. McGrath, M. Hickenbotham, et al. Dna sequencing of a cytogenetically normal acute myeloid leukaemia genome. *Nature*, 456(7218): 66–72, 2008.

87. H. Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013.

88. H. Li and R. Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

89. H. Li and R. Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

90. H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

91. H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

92. H. Li, J. Ruan, and R. Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome research*, 18(11):1851–1858, 2008.

93. H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, et al. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

94. K.-B. Li. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics*, 19(12):1585–1586, 2003.

95. N. Li and M. Stephens. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4): 2213–2233, 2003.

96. R. Li, Y. Li, K. Kristiansen, and J. Wang. Soap: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

97. R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang. SNP detection for massively parallel whole-genome resequencing. *Genome research*, 19(6):1124–1132, 2009.

98. R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang. SNP detection for massively parallel whole-genome resequencing. *Genome research*, 19(6):1124–1132, 2009.

99. R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

100. M. Litzkow, M. Livny, and M. Mutka. Condor-a hunter of idle workstations. In

Distributed Computing Systems, 1988., 8th International Conference on, pages 104–111. IEEE, 1988.

101. T. Magdon-Ismail, M. Nelson, R. Cheveresan, D. Scales, A. King, P. Vandrovec, and R. McDougall. Toward an elastic elephant enabling hadoop for the cloud. *VMware Tech. J*, 2013.

102. E. A. Manrao, I. M. Derrington, A. H. Laszlo, K. W. Langford, M. K. Hopper, N. Gillgren, M. Pavlenok, M. Niederweis, and J. H. Gundlach. Reading DNA at single-nucleotide resolution with a mutant MspA nanopore and phi29 DNA polymerase. *Nature biotechnology*, 30(4):349–353, 2012.

103. G. Marçais and C. Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.

104. E. R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends in genetics*, 24(3):133–141, 2008.

105. M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y.-J. Chen, Z. Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

106. A. Matsunaga and J. A. Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 495–504. IEEE Computer Society, 2010.

107. K. Megy, S. J. Emrich, D. Lawson, D. Campbell, E. Dialynas, D. S. Hughes, G. Koscielny, C. Louis, R. M. MacCallum, S. N. Redmond, et al. VectorBase: improvements to a bioinformatics resource for invertebrate vector genomics. *Nucleic acids research*, page gkr1089, 2011.

108. C. Meldrum, M. A. Doyle, and R. W. Tothill. Next-generation sequencing for cancer diagnostics: a practical perspective. *Clin Biochem Rev*, 32(4):177–195, 2011.

109. M. L. Metzker. Sequencing technologiesthe next generation. *Nature reviews genetics*, 11(1):31–46, 2010.

110. A. Mikhchi, M. Honarvar, N. E. J. Kashan, and M. Aminafshar. Assessing and comparison of different machine learning methods in parent-offspring trios for genotype imputation. *Journal of theoretical biology*, 399:148–158, 2016.

111. M. R. Miller, J. P. Dunham, A. Amores, W. A. Cresko, and E. A. Johnson. Rapid and cost-effective polymorphism identification and genotyping using restriction site associated dna (rad) markers. *Genome Research*, 17(2):240–248, 2007.

112. D. Money, K. Gardner, Z. Migicovsky, H. Schwaninger, G.-Y. Zhong, and S. Myles. LinkImpute: Fast and Accurate Genotype Imputation for Nonmodel Organisms. *G3: Genes, Genomes, Genetics*, 5(11):2383–2390, 2015.

113. G. E. Moore et al. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

114. C. Moretti, J. Bulosan, D. Thain, and P. J. Flynn. All-pairs: An abstraction for data-intensive cloud computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–11. IEEE, 2008.

115. D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand. CIEL: a universal execution engine for distributed data-flow computing. In *Proc. 8th ACM/USENIX Symposium on Networked Systems Design and Implementation*, pages 113–126, 2011.

116. D. E. Neafsey, G. K. Christophides, F. H. Collins, S. J. Emrich, M. C. Fontaine, W. Gelbart, M. W. Hahn, P. I. Howell, F. C. Kafatos, D. Lawson, et al. The evolution of the anopheles 16 genomes project. *G3: Genes— Genomes— Genetics*, 3(7):1191–1194, 2013.

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

118. K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 352–358. IEEE, 2002.

119. J. Reis-Filho. Next-generation sequencing. *Breast Cancer Res*, 11(Suppl 3):S12, 2009.

120. A. Rosset, L. Spadola, and O. Ratib. OsiriX: an open-source software for navigating in multidimensional DICOM images. *Journal of digital imaging*, 17(3): 205–216, 2004.

121. L. Salmela and E. Rivals. LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, page btu538, 2014.

122. E. E. Schadt, S. Turner, and A. Kasarskis. A window into third-generation sequencing. *Human molecular genetics*, 19(R2):R227–R240, 2010.

123. M. C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

124. P. Scheet and M. Stephens. A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and

haplotypic phase. *The American Journal of Human Genetics*, 78(4):629–644, 2006.

125. S. C. Schuster. Next-generation sequencing transforms today's biology. *Nature methods*, 5(1):16, 2008.

126. S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.

127. J. Shendure and H. Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

128. S. Shokralla, J. L. Spall, J. F. Gibson, and M. Hajibabaei. Next-generation sequencing technologies for environmental dna research. *Molecular ecology*, 21 (8):1794–1805, 2012.

129. C. C. Spencer, Z. Su, P. Donnelly, and J. Marchini. Designing genome-wide association studies: sample size, power, imputation, and the choice of genotyping chip. *PLoS Genetics*, 5(5):e1000477, 2009.

130. M. Stephens, N. J. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *The American Journal of Human Genetics*, 68(4):978–989, 2001.

131. Y. V. Sun and S. L. Kardia. Imputing missing genotypic data of single-nucleotide polymorphisms using neural networks. *European Journal of Human Genetics*, 16(4):487–495, 2008.

132. S. T O'Neil and S. J. Emrich. Haplotype and minimum-chimerism consensus determination using short sequence data. *BMC genomics*, 13(Suppl 2):S4, 2012.

133. H. Tao, D. R. Cox, and K. A. Frazer. Allele-specific KRT1 expression is a complex trait. *PLoS Genetics*, 2(6):e93, 2006.

134. J. F. Thompson and P. M. Milos. The properties and applications of single-molecule DNA sequencing. *Genome biology*, 12(2):217, 2011.

135. S. A. Tishkoff, E. Dietzsch, W. Speed, A. J. Pakstis, J. R. Kidd, K. Cheung, B. Bonne-Tamir, A. S. Santachiara-Benerecetti, P. Moral, M. Krings, et al. Global patterns of linkage disequilibrium at the CD4 locus and modern human origins. *Science*, 271(5254):1380–1387, 1996.

136. K. J. Travers, C.-S. Chin, D. R. Rank, J. S. Eid, and S. W. Turner. A flexible and efficient template format for circular consensus sequencing and SNP detection. *Nucleic acids research*, page gkq543, 2010.

137. O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.

138. R. Velasco, A. Zharkikh, J. Affourtit, A. Dhingra, A. Cestaro, A. Kalyanaraman, P. Fontana, S. K. Bhatnagar, M. Troggio, D. Pruss, et al. The genome of the domesticated apple (*Malus domestica Borkh*). *Nature genetics*, 42(10):833–839, 2010.

139. L. Wang, D. Chen, R. Ranjan, S. U. Khan, J. KolOdziej, and J. Wang. Parallel processing of massive EEG data with MapReduce. In *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pages 164–171. Ieee, 2012.

140. L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen. G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*, 29(3):739–750, 2013.

141. T. White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

142. C. J. Willer, S. Sanna, A. U. Jackson, A. Scuteri, L. L. Bonnycastle, R. Clarke, S. C. Heath, N. J. Timpson, S. S. Najjar, H. M. Stringham, et al. Newly identified loci that influence lipid concentrations and risk of coronary artery disease. *Nature Genetics*, 40(2):161–169, 2008.

143. D. Wu, C. M. Rice, and X. Wang. Cancer bioinformatics: A new approach to systems clinical medicine. *BMC bioinformatics*, 13(1):71, 2012.

144. L. Yu, C. Moretti, A. Thrasher, S. Emrich, K. Judd, and D. Thain. Harnessing parallelism in multicore clusters with the all-pairs, wavefront, and makeflow abstractions. *Cluster Computing*, 13(3):243–256, 2010.

145. M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *OSDI*, volume 8, page 7, 2008.

This document was prepared & typeset with pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2013[2013/04/16]) provided by Sameer Vijay and updated by Megan Patnott.*