Hummingbird: Efficient Performance Prediction for Executing Genomics Applications in the Cloud

Utsab Ray* Vandhana Krishnan† Amir Bahmani ‡ Cuiping Pan ** Keith Bettinger §
Philip Tsao ** Frank Mueller ¶ Michael Snyder ‖
† ‡ § ‖ Stanford Center for Genomics and Personalized Medicine, Stanford University, CA
‖ Department of Genetics, Stanford University, CA
** Epidemiology Research and Information Center for Genomics, VA Palo Alto, CA
* ¶ Department of Computer Science, North Carolina State University, Raleigh, NC
*† ‡ These authors contributed equally to this work.
* uray@ncsu.edu, † vandhana.krishnan@stanford.edu, ‡ abahman@stanford.edu, ¶ mueller@cs.ncsu.edu

***Abstract*—A major drawback of executing existing genomics pipelines on cloud computing facilities is that the onus of efficiently executing it on the best configuration lies on the user. Lack of knowledge regarding which cloud configuration is best to execute a pipeline often results in an unnecessary increase in cost due to selecting a more expensive cloud tier than needed. Resources in the cloud are expensive, so determining the best configuration before actually running the pipeline saves money and time. To this end, we introduce Hummingbird, a framework that predicts the best configuration to execute genomics pipelines on Google cloud.**

I. Introduction

Recently, the medical field has seen an increase in the availability of software to execute medical pipelines. Considering that medical pipelines handle large amounts of data and require significant amounts of time to execute, researchers have recently come to realize the benefit of executing medical pipelines on the cloud [7]. However, one of the main challenges with cloud computing is selecting an efficient and cost-effective set of resources, i.e, a configuration that utilizes as few resources as possible to deliver the output most efficiently. To this end, we propose Hummingbird, a framework that provides the user with the best configuration for executing a genomics pipeline.

Prior work has been done on investigating the usefulness of cloud computing for genomics applications. ODriscoll, et. al [11] and Lincoln Stein [12] highlighted the challenges associated with cloud computing and big data technologies in biology and genomics. In contrast, significant work has been done in predicting efficient configurations for different types of applications [14], [3], [8], [15], but no framework has specifically catered to genomics pipelines.

Hummingbird makes three major contributions:
(1) Downsampling: Hummingbird develops a technique to reduce the time required for the training phase prior to prediction. By reducing the size of the input files, Hummingbird can execute a genomics pipeline more quickly, thus decreasing the cost of training within our prediction framework.
(2) Prediction Model: Once downsampling is complete, Hummingbird uses the downsampled input files to run the entire pipeline on different types of Google cloud instances. Once the execution is complete, Hummingbird compares the execution time of all the instances and identifies three different configurations: "fastest", "cheapest", and "fast and cheap" (a compromise between getting the fastest execution time and the cheapest cost).
(3) Reduced Cost: By providing users with a fast-and-cheap configuration, Hummingbird helps users reduce costs while ensuring that their computation completes quickly.

II. Method

A. Overview of the Proposed System

Figure 1 depicts an overview of Hummingbird. In this section, we will review the various stages of the framework.

1) The user initially provides the pipeline he/she wishes to execute in the form of a configuration file.
2) Once Hummingbird extracts the different stage(s) of the pipeline from the configuration file, it searches a database to determine if configurations for that particular stage exist. If so, Hummingbird directly gives the user a list of configurations from which to choose. Otherwise, it proceeds with Stage 3.
3) Hummingbird then parses the entire user-provided configuration to obtain the information required to launch a cloud job. The user also must specify which stages of the pipeline support multi-threading. Once the names and locations of the input files are known, Hummingbird downsamples the input files. The details of downsampling are given in Section II-B.
4) When downsampling is complete, Hummingbird runs the pipeline with the downsampled input files.
5) Once the pipeline has finished executing, Hummingbird calculates the cheapest, fastest, and the fast-and-cheap configurations based on the measured execution times. Explanation of the different configurations are provided in Section III.

B. Downsampling Design

One of the main contributions of Hummingbird is to downsample the input files so that the entire pipeline can be executed quickly, thus reducing both the execution time and the cost of the training phase. As we focus on genomics applications, Hummingbird downsamples inputs of genomics

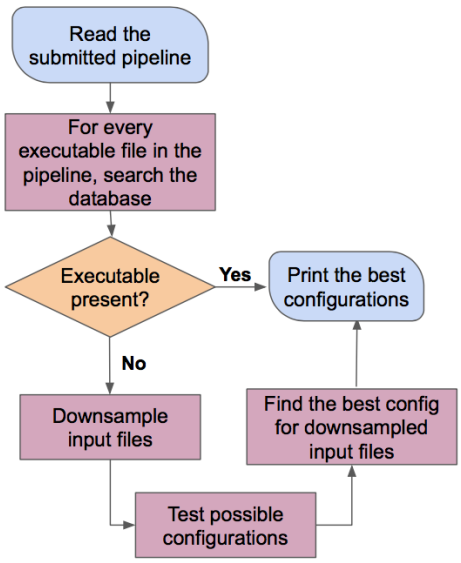

Fig. 1: An Overview of Hummingbird

applications systematically. The two data formats most common in genomics applications are *fastq.gz* and *BAM*.

Any fastq.gz inputs are truncated, i.e., only the first n lines comprise the downsampled file. Downsampling the file to one million lines was empirically determined to provide the best result as it drastically reduced the time required for training while still following the same trends as pipeline executions on the full input. All fastq files contain 4 lines per sequence. Since sequences are not dependent on one another, we can remove any number of lines while downsampling, without affecting the correctness of the output and the behavior of the application.

BAM files are another way of storing biological sequences, just like fastq files. BAM files have a header and an alignment section. The alignment section has 11 mandatory fields. These files are downsampled using the *DownsampleSam* tool available in the GATK docker image. A separate cloud job must be launched just for downsampling the BAM files. The amount of downsampling can be controlled by the probability (P) argument. A P of 0.1 indicates ten percent of the reads in the BAM file will be kept in the downsampled file, thus reducing the size of the BAM file tenfold. Just like fastq files, the alignment section of BAM files is not dependent on other alignment sections, which facilitates downsampling.

III. Experimental Framework

Pipelines: Two of the most popular variant discovery workflows in genomics are part of the GATK [10] best practices provided by the Broad Institute [13]. For our experiments, we selected GATK 3.7 that uses HaplotypeCaller to call germline SNPs (single nucleotide polymorphisms) and indels (insertions-deletions). The popular pipeline MuTect2 is a somatic SNP and indel caller that fuses the somatic genotyping engine of the original MuTect [5] with aspects of the GATK HaplotypeCaller algorithm. Also, the pipeline that employs MuTect2 overlaps significantly with the one that uses HaplotypeCaller. These variant calling pipelines can be run on the cloud or on a high-performance computing cluster. However, each step in any of the above variant calling pipelines utilizes different software tools, and optimizing the corresponding computational resources can save time and costs. We illustrate next how our Hummingbird tool provides suggestions on which computational resources to use if running on Google cloud.

Datasets: For the GATK pipeline, we obtained the open-source Illumina Platinum Genomes [6], which are available on public Google cloud buckets [1]. The reference file we utilized was *GRCh37-lite*, also available publicly in Google cloud. For MuTect2, we used the bam files provided by the Texas Cancer Research Biobank [4].

Cloud job submission via dsub: Hummingbird leverages Google cloud as the cloud service provider. *dsub* is a command-line tool to submit jobs on Google cloud. Submitting jobs with dsub requires the project name, the zone in which the virtual machine (VM) is to be launched, input bucket, output bucket, logging bucket, docker image name, machine configuration and a command/program to execute. Upon receiving this information, Google cloud executes that command on a docker image launched on a VM with the machine configuration specified by dsub.

Google cloud: Google cloud offers a wide variety of instances with different costs, which can be separated into three main categories: high-CPU, standard and high-mem. The CPU platform is the same across all three categories, but the amount of main memory (RAM) available to a VM differs. Table I indicates the instance types used in our experiments. We consider high-CPU, standard and high-mem with 8, 16 and 32 VCPUs due to our limited budget as considering all instance types would result in an excessive number of permutations with high cost. Of course, by design, Hummingbird could execute all instance types available in Google cloud, i.e., all permutations of configurations.

TABLE I: Google Cloud Instance Types

Machine Type	VCPUs	Memory(GB)	Price(hourly)
n1-highcpu-8	8	7.20	$0.2836
n1-highcpu-16	16	14.40	$0.5672
n1-highcpu-32	32	28.80	$1.1344
n1-standard-8	8	30	$0.3800
n1-standard-16	16	60	$0.7600
n1-standard-32	32	120	$1.5200
n1-highmem-8	8	52	$0.4736
n1-highmem-16	16	104	$0.9472
n1-highmem-32	32	208	$1.8944

IV. Experimental Results and Analysis

Once the input files have been downsampled, Hummingbird runs the pipeline using those reduced inputs on three different instance types (VCPU counts) for each of the three different categories (high-CPU, standard and high-mem). Once each execution time is recorded, a table is created for each instance category that summarizes speedup, scaling and cost concisely. An abstract summary of these metrics is depicted in Table II. The first column indicates the number of VCPUs on each instance. T_8 indicates an instance with 8 VCPUs, and so on. Ideal Speedup is the speedup that the instance would show under perfect scaling without resource sharing and is calculated as the ratio of the number of VCPUs in the current instance over the number of VCPUs in the base instance. In practice, speedup does not necessarily follow the ideal curve. To this end, the real speedup is calculated as the ratio of execution

time of the base instance over the execution time of the current instance. Since the real speedup differs for each instance, we further normalize the speedup in the next column to facilitate a comparison between all instances. The normalized speedup is used to determine the instance with the most benefit and how many resources should be committed before a given pipeline stops scaling.

TABLE II: Decision Table

Machine	Ideal Speedup	Real Speedup	Execution Time	Normalized Speedup	Cost
T_8(base)	1	S_8=T_8/T_8	E_8	1	C_8
T_{16}	2	S_{16}=T_8/T_{16}	E_{16}	S_{16}	C_{16}
T_{32}	4	S_{32}=T_8/T_{32}	E_{32}	S_{32}/2	C_{32}

Hummingbird recommends three different configurations to the user:

• Cheapest: Hummingbird chooses the cheapest configuration over the three categories and then selects the instance type (number of VCPUs) one with the lowest cost.

• Fastest: Hummingbird chooses the fastest configuration over the three categories and then selects the instance type with least execution time.

• Fast and Cheap: Hummingbird chooses the category with the highest normalized speedup, i.e., the instance which scaled best, and then selects the instance type with the lowest cost.

GATK: Hummingbird provides a stage-by-stage recommendation for every pipeline. This is better than one recommendation for the entire pipeline because what might be fast or cheap for one stage may not be fast or cheap for another stage.

Tables III- VII reflect Hummingbird′s recommendation vs. the recommendation populated by executing that stage of the pipeline on the whole input, recording the execution time, and then manually calculating the cheapest, fastest, and fast-and-cheap configurations.

1) BWA: BWA (Burrows-Wheeler Aligner) [9] is a fast, short read aligner. For our experiments, we chose BWA-MEM because it is widely used for sequence alignment and is also recommended in the GATK best practices for both germline and somatic variant calling pipelines. Table III indicates that Hummingbird predicts the cheapest configuration correctly for BWA. It also recommends highcpu-32 to be the fastest, whereas the actual fastest configuration is standard-32, which is 3.1% faster than highcpu-32. For the fast-and-cheap, Hummingbird recommends standard-16 while standard-8 would have been the correct choice according to the table. Hummingbird results in 5.1% higher cost and 47% faster execution, which is actually a good choice.

TABLE III: BWA Results

	Hummingbird		Whole Input	
	Config	Exec Time (s)	Config	Exec Time (s)
Cheapest	standard-8	26.075	standard-8	92,124.358
Fastest	highcpu-32	15.757	standard-32	30,286.578
Fast & Cheap	standard-16	19.094	standard-8	92,124.358

2) Picard: Picard [2] is a set of command line tools (in Java) for manipulating high-throughput sequencing (HTS) data and formats such as SAM/BAM/CRAM and VCF. We used two different Picard tools in our GATK pipeline, *SortSam* (Picard Stage 1 and 3) and *MarkDuplicates* (Picard Stage 2). The latter locates and tags duplicate reads in a BAM or SAM file. Table IV depicts the results for the first stage, where the Picard tools are used. For Picard Stage 1, Hummingbird correctly predicts highmem-32 as the fastest. The cheapest prediction by Hummingbird is 32.9% more costly than the actual cheapest one. The fast-and-cheap prediction is -2% faster but 63.5% higher in cost than the correct choice for whole input.

TABLE IV: Picard Stage 1 Results

	Hummingbird		Whole Input	
	Config	Exec Time (s)	Config	Exec Time (s)
Cheapest	standard-8	10.363	highcpu-8	26,015.502
Fastest	highmem-32	9.187	highmem-32	24,292.8
Fast & Cheap	highmem-8	10.157	highcpu-8	26,015.502

3) BQSR: The GATK base quality score recalibration (BQSR) is a data pre-processing step that detects systematic errors made by the sequencer when it estimates the quality score of each base call. The two stages of BQSR use *BaseRecalibrator* and *PrintReads* for the first and second stages, respectively. Table V shows that Hummingbird correctly predicts two of three configurations for stage 1 of BQSR. The fastest prediction is 7.3% slower than the actual fastest one.

TABLE V: BQSR Stage 1 Results

	Hummingbird		Whole Input	
	Config	Exec Time (s)	Config	Exec Time (s)
Cheapest	standard-8	776.193	standard-8	29,003.23
Fastest	highcpu-32	608.741	highmem-32	15,944.26
Fast & Cheap	standard-16	751.42	standard-16	20,113.56

4) HaplotypeCaller: The GATK HaplotypeCaller cite-GATK is capable of calling SNPs and indels simultaneously via local de-novo assembly of haplotypes in an active region. Table VI shows that Hummingbird predicts two configurations correctly for Haplotype. The fast-and-cheap prediction is 37.7% lower in cost but 24.5% slower.

TABLE VI: Haplotype Results

	Hummingbird		Whole Input	
	Config	Exec Time (s)	Config	Exec Time(s)
Cheapest	standard-8	2,306.35	standard-8	73,105.23
Fastest	highmem-32	2,069.97	highmem-32	43,179.96
Fast & Cheap	standard-8	2,306.35	standard-16	58,676.61

MuTect2: Mutect2 [5] has the ability to call short somatic SNPs and indels in a tumor-normal pair. In Mutect2, using a normal sample or a panel of normals helps filter germline variants specifically. Table VII shows that Hummingbird predicts all three configurations correctly for MuTect.

TABLE VII: MuTect2 Results

	Hummingbird		Whole Input	
	Config	Exec Time (s)	Config	Exec Time (s)
Cheapest	highcpu-8	1,348.591	highcpu-8	40,754.48
Fastest	highmem-32	957.078	highmem-32	13,674.71
Fast & Cheap	highcpu-16	1,164.396	highcpu-16	24,087.63

Downsampling: The downsampling concept can theoretically be applied to other pipelines. The only caveat is that the input file itself must not have dependencies. E.g., if we

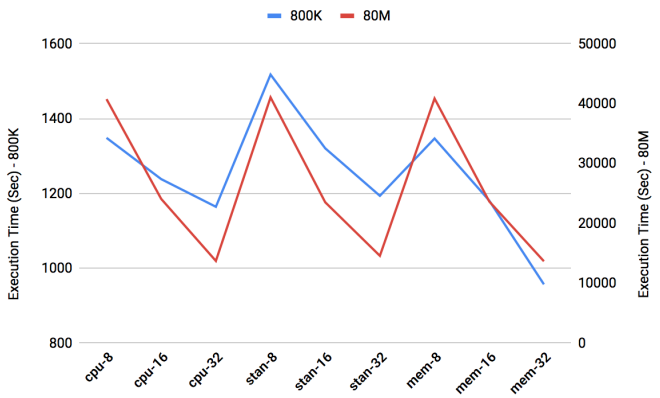

Fig. 2: Downsampling for MuTect2

remove half of the input file and execute the pipeline on the remaining half, the execution should follow the same trends as the execution on the whole input. If trends were to differ across inputs, one could not conclusively determine which instance is the cheapest or fastest.

Figure 2 depicts the comparison between execution on the downsampled input (blue, left y-axis) and execution on all the input files (red, right y-axis) for MuTect2. The BAM file we experimented with contained approximately 80 million lines, which were downsampled to $800k$ lines for faster training of Hummingbird. The figure indicates that the execution times for the full and the downsampled inputs follow the same trends. This facilitates predictions based on executing the downsampled files on the entire pipeline.

V. Related Work

Prior work exists on predicting the best configuration to run an application on the cloud, but none of them provide a cost effective prediction for genomics frameworks. While Ernest [14] and CherryPick [3] (and others) work on a variety of pipelines from different fields, they require extensive configurations. Ernest reports results for genomics pipelines such as ADAM and GenBase. However, Ernest requires knowledge about which data points are to be collected, i.e., knowledge about the input file structure. In contrast, Hummingbird is capable of selecting the datapoints to run the entire pipeline on. CherryPick employs Bayesian optimization to find the best configuration. CherryPick could in principle be applied to genomics pipelines, but does not report any such results. However, CherryPick also requires extensive configurations and multiple steps to find the best configuration. This increases both training time and training cost. None of these prior approaches use downsampling.

VI. Discussion

Hummingbird provides a range of configurations that users can choose from according to their requirements. In this paper, we compared Hummingbird′s results with those obtained by executing the application on the whole input file and found that Hummingbird was able to predict the best configuration in many cases. Some mispredictions are off by only a few percent or may even be a better trade-off between cost and speed while others diverge more significantly, which indicates that our heuristics could be improved (subject to future work).

To date, we have only tested genomics applications. By design, Hummingbird can be used for other cloud applications as well. We currently vary the number of VCPUs and the amount of main memory and suggest a best configuration. In the future, we plan to tune other factors so that we can provide the user with an even better optimized solution.

VII. Conclusion and Future Work

Hummingbird has the ability to downsample inputs and provides an efficient and effective prediction model based on a much reduced cost of running the pipeline for training. We have shown the benefit of downsampling inputs and that ultimately aided in recommending different configurations to the user for execution of a genomics pipeline.

Acknowledgment

In memory of Someyra Nazemi Gelian, a graduate student at NC State who battled to the end with cancer and ultimately inspired this work. This work was partially funded by a gift from the Stanford Data Science Initiative (https://sdsi.stanford.edu/).

References

[1] Illumina platinum genomes. https://console.cloud.google.com/storage/browser/genomics-public-data/platinum-genomes.

[2] The picard command-line tools. https://broadinstitute.github.io/picard/.

[3] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*, volume 2, pages 4–2, 2017.

[4] Lauren B Becnel, Stacey Pereira, Jennifer A Drummond, Marie-Claude Gingras, Kyle R Covington, Christie L Kovar, Harsha Vardhan Doddapaneni, Jianhong Hu, Donna Muzny, Amy L McGuire, et al. An open access pilot freely sharing cancer genomic data from participants in texas. *Scientific data*, 3:160010, 2016.

[5] Kristian Cibulskis, Michael S Lawrence, Scott L Carter, Andrey Sivachenko, David Jaffe, Carrie Sougnez, Stacey Gabriel, Matthew Meyerson, Eric S Lander, and Gad Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature biotechnology*, 31(3):213, 2013.

[6] Michael A Eberle, Epameinondas Fritzilas, Peter Krusche, Morten Källberg, Benjamin L Moore, Mitchell A Bekritsky, Zamin Iqbal, Han-Yu Chuang, Sean J Humphray, Aaron L Halpern, et al. A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. *Genome research*, 2016.

[7] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 460–469. ACM, 2010.

[8] Chin-Jung Hsu, Vivek Nair, Vincent W Freeh, and Tim Menzies. Low-level augmented bayesian optimization for finding the best cloud vm. *arXiv preprint arXiv:1712.10081*, 2017.

[9] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*, 2013.

[10] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 2010.

[11] Aisling ODriscoll, Jurate Daugelaite, and Roy D Sleator. big data, hadoop and cloud computing in genomics. *Journal of biomedical informatics*, 46(5):774–781, 2013.

[12] Lincoln D Stein. The case for cloud computing in genome informatics. *Genome biology*, 11(5):207, 2010.

[13] Geraldine A Van der Auwera, Mauricio O Carneiro, Christopher Hartl, Ryan Poplin, Guillermo Del Angel, Ami Levy-Moonshine, Tadeusz Jordan, Khalid Shakir, David Roazen, Joel Thibault, et al. From fastq data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. *Current protocols in bioinformatics*, 43(1):11–10, 2013.

[14] Shivaram Venkataraman, Zongheng Yang, Michael J Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *NSDI*, pages 363–378, 2016.

[15] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 452–465. ACM, 2017.