Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Subject Section

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

Amir Bahmani [*,1,2,3], **Ziye Xing** [*,2,3], **Vandhana Krishnan** [*,2,3], **Utsab Ray** [*,5], **Frank Mueller** [5], **Amir Alavi** [1], **Philip S. Tsao** [4], **Michael P. Snyder** [1,2,3], **Cuiping Pan** [4]

[1]Stanford Healthcare Innovation Lab, Stanford University, CA
[2]Stanford Center for Genomics and Personalized Medicine, Stanford University, CA
[3]Department of Genetics, Stanford University, CA
[4]Palo Alto Epidemiology Research and Information Center for Genomics, VA Palo Alto, CA and
[5]Computer Science Department, North Carolina State University, Raleigh, NC

*These authors contributed equally to this work.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: A major drawback of executing genomic applications on cloud computing facilities is the lack of tools to predict which instance type is the most appropriate, often resulting in an over- or under- matching of resources. Determining the right configuration before actually running the applications will save money and time. Here, we introduce Hummingbird, a tool for predicting performance of computing instances with varying memory and CPU on multiple cloud platforms.
Results: Our experiments on three major genomic data pipelines, including GATK HaplotypeCaller, GATK MuTect2, and ENCODE ATAC-seq, showed that Hummingbird was able to address applications in command line specified in JSON format or workflow description language (WDL) format, and accurately predicted the fastest, the cheapest, and the most cost-efficient compute instances in an economic manner.
Availability: Hummingbird is available as an open source tool at:
https://github.com/StanfordBioinformatics/Hummingbird
Contact: mpsnyder@stanford.edu, cuiping@stanford.edu
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Biomedical research has seen an increasing adoption of public cloud platforms, owing to their capabilities in handling ever larger amounts of data and more complex pipelines (Gunarathne *et al.*, 2010; O'Driscoll *et al.*, 2013). Inherent to this data deluge is the cost associated with computation. For example, in genomics, an important field within biomedical research, the costs of data computing, together with storage will soon outweigh the cost of genome sequencing (Stein, 2010; Van der Auwera *et al.*, 2013). Careful selection of resource-matching compute instance types can significantly reduce computational costs. While public cloud platforms provide a wide range of instance types with different combinations of memory capacities and central processing units (CPU), it remains a big challenge to choose the configuration that will perform the task in an efficient and economic manner.

Multiple tools exist for predicting the best configuration for running various applications in the cloud. However, they require extensive setup and are not specifically designed for genomic pipelines (Alipourfard *et al.*, 2017; Hsu *et al.*, 2017; Yadwadkar *et al.*, 2017). Recently, Ernest (Venkataraman *et al.*, 2016) reported optimizing configurations for genomic pipelines, such as ADAM and GenBase, but their method required prior knowledge of input file structure, and, notably, it addressed tools specifically based on the Apache Spark framework. For commonly used genomic workflows built outside of Apache Spark, such as BWA and Bowtie2, tools for optimizing configuration are needed.

Here, we present Hummingbird, a performance predictor for identifying the best instance type for executing genomic pipelines in public cloud platforms. We address performance requirements by measuring

the shortest runtime, the lowest cost, and the most efficient compute expenditure (i.e., most cost-efficient). Our method can be applied to run on multiple cloud platforms, including but not limited to, Google Cloud Platform (GCP), Amazon Web Services (AWS) and Microsoft Azure. In addition, Hummingbird accommodates various types of bioinformatics pipelines, making it generalizable for finding the best instance configuration for a broad spectrum of genomic applications.

2 Results

2.1 Architectural Design of Hummingbird

Hummingbird is designed to predict the cheapest, fastest, and most cost-efficient machine types for genomics applications in the cloud (Figure 1). It parses a pipeline configuration file supplied by the user to extract essential parameters and then runs the pipeline accordingly on user-selected cloud instance types while profiling application behavior. Specifically, Hummingbird utilizes a component called *Memory Profiler* to quickly identify instance types with inadequate memory and eliminates them from further testing. For the remaining instance types, each with a unique combination of memory and CPUs, Hummingbird compares their runtimes and costs via performance profiling in the *Recommendation Engine* component and recommends the best instance types for each performance category.

We devised a few techniques to accelerate and reduce the cost of these processes, which is described in the following two sub-sections namely 2.2 and 2.3.

2.2 Memory Profiler prunes out resource inadequate instances

Inadequate memory is often the cause of pipeline failure. To address this, we devised the Memory Profiler component in Hummingbird to first focus on the instance family with the largest memory resources, e.g., the high-mem instances in GCP, the R5 instances in AWS or the Standard E2−32 instances in Azure. Hummingbird measures the memory usage in these instance types. Besides accepting whole input data, we also provide a prediction method based on downsampling in order to make the process more economic. This cost-effective prediction strategy randomly downsamples whole input data in the fractions of 0.1%, 1% and 10%, runs them through the instances, and extrapolates the memory usage for the whole input using linear regression

$$(memory \sim \log_{10}(downsampling\ ratios))$$

The predicted memory usage is then used to prune instance types that do not fulfill this resource requirement. Instance types that have a memory above this threshold will then continue with performance profiling in the Recommendation Engine.

2.3 Recommendation Engine measures performance of instance types

Hummingbird uses 10% randomly downsampled input to generate performance profiles. Runtimes are recorded for each instance type and converted to cost and cost-efficiency for comparison (see Methods). In addition to this default downsampling ratio, we also implemented a dynamic downsampling approach for achieving a potentially faster and more cost-effective turnover. This method starts with running the pipeline on a 0.1% dataset randomly sampled from the whole input. Runtime reduction with respect to increased amount of virtual CPUs (vCPU, equal to a thread) is calculated, resulting in speedup efficiency (refer to Methods). If speedup efficiency across any of the tested instances

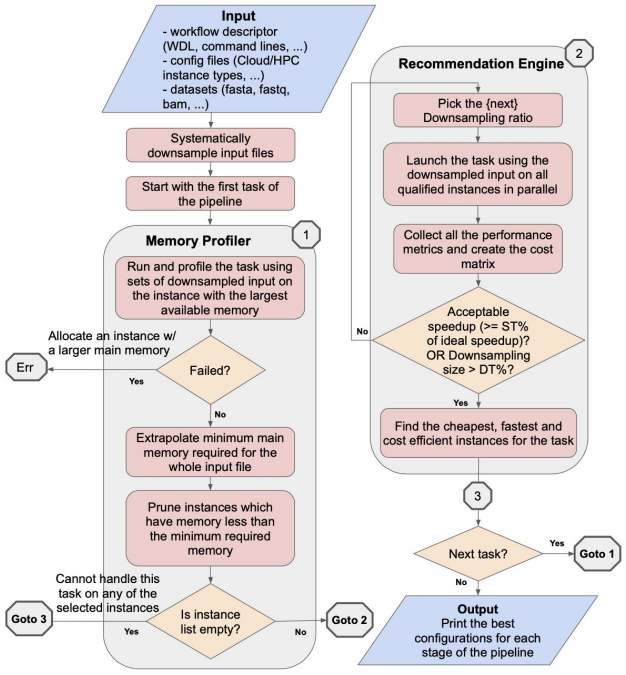


Fig. 1. Hummingbird predicts the fastest, cheapest, and most cost-efficient compute instances for genomics applications in the cloud. The complex components of Hummingbird are the Memory Profiler and Recommendation Engine, denoted in the figure as (1) and (2), respectively. The Memory Profiler analyzes memory usage to prune out instance types with insufficient memory resources. The Recommendation Engine compares performance metrics among compute instances so as to recommend the best performing machine types. In both of these Hummingbird's complex components, predictions are based on downsampled datasets to reduce costs. After the processes linked to the Hummingbird's two components are completed for each task in the genomics application, Hummingbird moves on to the next task, as indicated by (3) in the figure. Then, for this new task Hummingbird reiterates through the components of Memory Profiler and Recommendation Engine. This cycle repeats until all tasks in the pipeline are exhausted. Subsequently, Hummingbird returns the fastest, the cheapest, and the most cost-efficient computing instances for each task in the genomics pipeline.

exceeds the threshold, e.g., $\frac{2}{3}$, then the profiling stops, and all performance metrics, such as runtime, cost, and cost efficiency, are subjected to final analysis. Alternatively, Hummingbird will move on to the next round of profiling by increasing the input to 1% randomly sampled whole input and recalculating the speedup efficiency. Hummingbird will iterate this process with up to three rounds that involves a downsampling limit of 10% of the whole input to identify the best-performing instances. Eventually, the best configurations derived from the last round are recommended to the user.

To maximize the benefit of Hummingbird, it is suggested to present the pipeline by stages, so that recommendation for every stage of the pipeline can be obtained.

2.4 Applications

We demonstrate the utility of Hummingbird via three genomics pipelines: the Genome Analysis Toolkit (GATK) HaplotypeCaller for detecting germline DNA variants (McKenna *et al.*, 2010), the GATK MuTect2 for detecting somatic DNA variants (Cibulskis *et al.*, 2013), and an ATAC-seq pipeline developed in the ENCODE project for assessing genome-wide chromatin accessibility (Davis *et al.*, 2018). These pipelines represent the most popular and comprehensive data processing pipelines in genomics, each composed of multiple computational steps. In addition, while the two

GATK pipelines were written in scripts of command lines, the ATAC-seq pipeline was specified in workflow description language (WDL) (`https://openwdl.org/`) and its jobs are managed by a workflow management system called Cromwell (Voss *et al.*, 2017) (`https://github.com/broadinstitute/cromwell`). We show that Hummingbird is capable of tackling applications with varying complexities.

We profiled the three genomics pipelines on GCP through Hummingbird using the downsampling technique as described. To derive the ground truth, we also ran the whole input data in all instance types included in our experiments (Supplemental Tables 1-3), which served as validation for Hummingbird's prediction results. If the bioinformatics tools used by the pipelines supported multithreading, we turned it on to maximize their performance. Each experiment was repeated three times and the averages were reported. Collectively, there were 16 computational steps in the three pipelines. As five data processing steps were common to both the GATK HaploytypeCaller and Mutect2 pipelines, we analyzed them jointly and called them the GATK pipelines. We note that although different input datasets were used for profiling these two GATK pipelines, e.g., a genome used for HaplotypeCaller versus a tumor-normal pair used for Mutect2, this should not affect the performance evaluation as these datasets represent typical genomics datasets.

Overall, Hummingbird achieved a prediction accuracy of 85.4%, measured by the matching between the predicted instance types and the ground-truth best instance types (Figure 2). Predictions for the fastest instances were slightly worse compared to that for the cheapest instances or the most cost-efficient instances. Particularly for the GATK pipelines, this prediction always resulted in a longer runtime, although the highest ones were still within the boundary of 20% error rate (Supplemental Figure 1). Prediction for the fastest instance for the ATAC-seq pipeline was significantly better (Supplemental Figure 2), indicating there was no systematic bias in the method for the runtime prediction. We note that most of these runtime differences were minor and under 10 minutes. Therefore, for practical reasons, we treated the predicted optimal instance as equivalent to the ground truth optimal instance as long as the error rate was within 10%. However, the error rates are labeled on top of each cell in Figure 2 and fully revealed.

When examined in detail on how efficiently these predictions could be achieved, we found that the Memory Profiler component had a large effect. For 12 of the 16 computational steps in the three pipelines, instance pruning removed between 11% to 50% of instance types from subsequent analysis, demonstrating that removal of resource-inadequate instance types greatly narrows the scope of search. Note that in our experiments we pruned instances based on memory for the whole input inferred from systematically downsampled datasets. This modeling using linear regression, though basic, achieved an error rate bounded at 20% for many steps (Supplemental Figures 3-5). This approach of running on downsampled datasets, while achieving a reasonable prediction accuracy, sped up the process and reduced costs. We leveraged this approach for both memory profiling and in validation via performance profiling. While the Memory Profiler used 0.1%, 1%, and 10% downsampled datasets, the Recommendation Engine used 10% of the randomly downsampled data as a default. The total cost of these memory profiling steps for the three pipelines (Supplemental Tables 4a and 6a, with each experiment repeated three times, was only a fraction of that for running the whole inputs (Supplemental Tables 4b and 6b). After the memory profiling, we take note of the instance types that passed the full memory thresholds for each stage in a pipeline. The subsequent performance profiling is only performed on these selected instances, instead of the entire family of instances. Since this reduces the number of jobs launched, it saves the user from incurring added costs during performance profiling.

With the prediction results of Hummingbird, we now gained a better understanding of how carefully choosing instance types could largely impact the runtimes and costs (Figure 3). For example, running the NA12890 genome on a set of fastest instances (i.e., speed-optimized) recommended by Hummingbird can complete the GATK germline variant calling in approximately 54 hours. However, if time is not the critical factor and another 12 hours are permitted, the GATK HaplotypeCaller can be run on the cost-optimized instances and, in return, the cost will reduce by 40%. A similar pattern was observed for the ATAC-seq pipeline and Mutect2 pipelines. Given these large differences, we conclude that it is necessary and critical to profile pipeline performance on different instance types so as to optimize for the data processing requirements. Particularly for large scale data processing involving enormous volumes of data, the cumulative differences could be paramount.

2.5 A cloud agnostic solution

To extend its utility in other cloud platforms, we implemented Hummingbird in AWS and Azure Cloud, and carried out a proof-of-concept study on BWA-MEM (Li, 2013). This is a tool for aligning DNA sequencing reads to the reference human genome, a critical step in the variant calling process. We randomly downsampled the sequence reads of the NA12890 genome to 0.1%, 1% and 10%, and used Hummingbird to profile the performance of running them as well as the whole input in two families of memory-optimized instances in each cloud platform (Supplemental Tables 1−3). For comparison, we extracted runtimes for the same inputs on GCP from our previous experiments on the GATK HaplotypeCaller pipeline. Although runtimes were different for the three cloud platforms owing to different instance configurations, an obvious reduction pattern with doubling compute resources was observed within each family of instances (Supplemental Figure 6). In addition, speedup efficiency was strong for nearly all instances tested (Supplemental Table 8). We compared the best instance types recommended by Hummingbird from the three cloud platforms (Supplemental Table 11) and found them to utilize very similar computational resources, suggesting Hummingbird functioned well regardless of the cloud platforms. Based on these experiences, we provide a guideline for further extending Hummingbird to other cloud services.

3 Discussion

In this paper, we presented Hummingbird, an efficient tool for predicting performance of compute instances for various genomic applications in the cloud. We demonstrated its utility in finding the best performing instances for three comprehensive genomic pipelines, including the GATK germline and somatic variant calling pipelines, and the ENCODE chromatin accessibility detection pipeline for ATAC-seq. We analyzed its utility on GCP, and showed that it could be extended to other cloud platforms such as AWS and Azure, rendering it agnostic to cloud platforms. Hummingbird returns the best instance types that offer the fastest runtime, the lowest cost, and the most cost-efficient services. Our examination revealed that it achieved an overall prediction accuracy of 85.4%. The adoption of downsampling methods combined with the inherent feature of parallel computation in the cloud has made Hummingbird an efficient and economical tool for performance prediction. The total runtime and cost of the memory and performance profiling, both based on downsampled datasets, were only a fraction of that for running the whole input data. Moreover, we obtained significant benefits of using the instance types recommended by Hummingbird, as observed by the differences in runtimes and costs between the instances optimized for different purposes. In some cases these differences were several-fold larger. Note that this comparison was done between Hummingbird recommended instances, targeted either at speed or cost; without further optimization effort, the gap could grow larger.

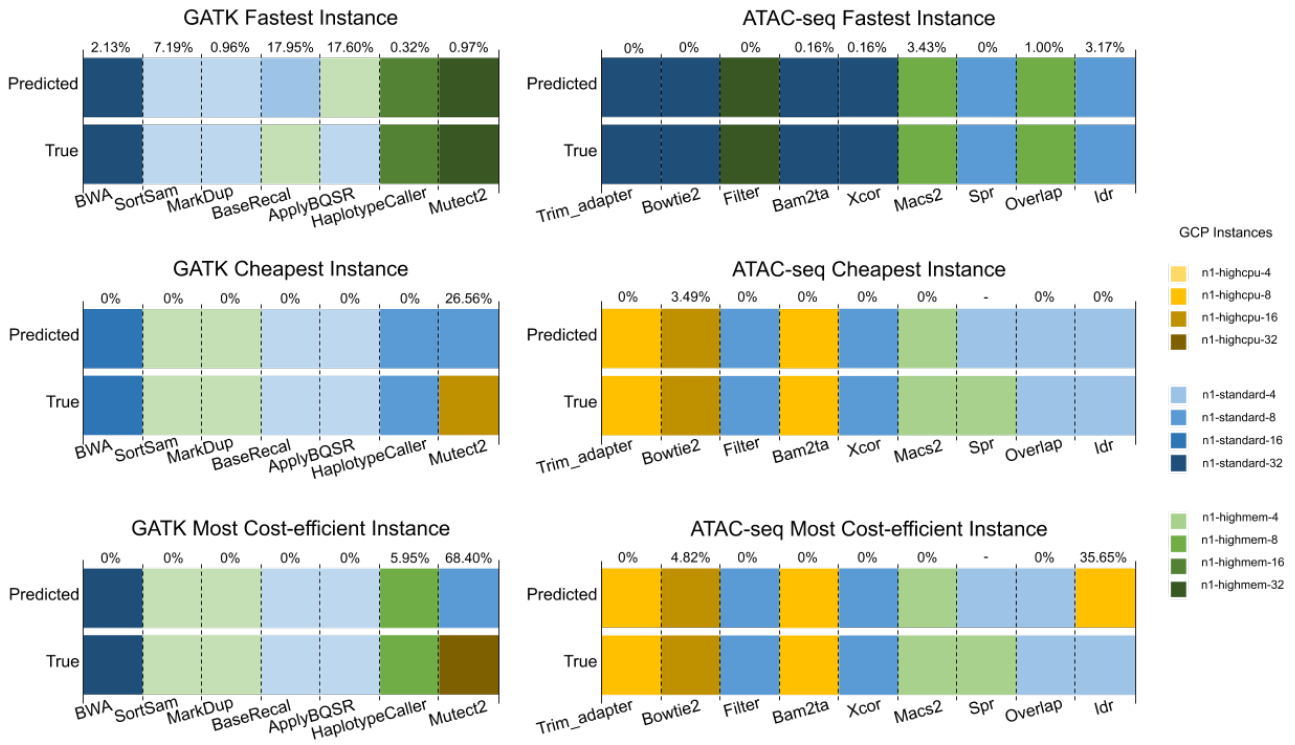


Fig. 2. Prediction accuracy of Hummingbird in selecting the best performing instances by measure of runtime, cost, and cost efficiency. The GATK pipelines are listed in the left column, and the ATAC-seq pipeline is on the right. In each performance category, the predicted optimal instance for each pipeline step is listed in the upper row, and the ground-truth optimal instance is listed in the lower. The instance types are represented by different color cells. Prediction error is indicated by the percentage difference on top of each cell. For instance types where the difference was within 10% error rate, we treat the predicted instance the same as the ground truth instance but also indicate the difference. A failed run on Spr step was observed with the predicted full memory, and therefore indicated by "$-$" in the figure.

Leveraging the fault-tolerant features of cloud computing techniques, it handles network errors and allows multiple attempts to rerun tasks. Hummingbird also harbors granularity in its profiling function. With pipelines written or annotated in clear modules, it can parse and extract each step of the pipeline and identify not only the best compute instance(s) for the full pipeline but also for individual steps. The bioinformatics pipelines used in our experiments involved sophisticated computational steps and diverse programming languages. While some pipelines were presented as scripts of original command lines, others were packaged in WDL. For the former, annotation was added to the configuration JSON file for Hummingbird to extract stages of the pipeline, and accordingly profiled the individual stages. For the latter, Hummingbird used Cromwell for parsing the workflow and subsequent execution of the stages.

Hummingbird accommodates input files of formats other than BAM and FASTQ and this is possible by skipping the downsampling step which is a caveat in terms of costs. However, the absence of the downsampling step provides the user the flexibility to execute pipelines that require varied input file formats (e.g., BED, VCF, CRAM). We plan to incorporate downsampling of the above input file formats in the future. Hummingbird's GitHub page has examples of tools that perform conversions to different formats that a user could utilize to run this framework.

Hummingbird is currently able to parse workflows written in WDL using Cromwell. Hummingbird can potentially work with other workflow managers as well, and that is something we plan to add in future versions. Hummingbird can utilize a broad spectrum of bioinformatics tools and recommend machine types in categories desired by a user. It owes much of this generality to the container technology in the cloud, which packages an application to run with isolated dependencies and meanwhile achieves consistency. In the configuration file, the user can supply custom Docker images for each stage of the pipeline or a single Docker image for the entire pipeline.

Hummingbird does not restrict itself to specific workflows. However, to utilize its downsampling feature, downsampling methods compatible with workflow input files need to be included. We incorporated methods for downsampling BAM and FASTQ files for our experiments with the three genomic pipelines. Similarly, additional downsampling methods can be incorporated to address the need of specific workflows. For this, we provided a guideline on the GitHub site. In addition, we provided examples of tools that help convert between various file formats.

Although the current methods implemented in Hummingbird functioned well to a large degree, we acknowledge that prediction based on downsampled datasets, while drastically reducing costs, can introduce limitations. For example, memory profiling for the Spr step in the ATAC-seq pipeline underestimated memory for the whole input by 80%, which consequently included two instance types with insufficient memory capacity, and eventually failed the runs with the whole input. This discrepancy was due to the Spr step requiring a linear increase of memory with data size. When applying a linear regression to the original values $(memory \sim (downsampling\ ratio))$ rather than the logarithmic transformed values $(memory \sim \log_{10}(downsampling\ ratios))$, we obtained an almost 100% matched prediction. This was an exception, as the remaining tools in the three genomics pipelines fit better with the model of logarithmic transformation, indicating that they use memory resources more efficiently. Although we expect the former case to be less common and therefore have set the logarithmic transformation as default, we provide other options in Hummingbird for cases where the recommended instance type fails: (1) switching to the model of linear regression to original values for memory prediction, (2) adding a certain

percentage of buffering memory on top of the predicted full memory, or (3) running memory profiling directly on the whole input data. These options enhance Hummingbird's functionality.

Even though we designed a dynamic downsampling approach for performance profiling, we found that of the 16 steps in the three pipelines, only BWA-MEM effectively used this strategy. BWA-MEM showed sufficient speedup efficiency in all instance types even for small datasets, and therefore justified the extrapolation we obtained for the whole input in performance evaluation. We found that for runs on the smallest dataset tested, 0.1% randomly downsampled from a NA12890 genome of 707 million reads, one can accurately predict the performance behavior of the whole input. However, for the other 15 steps of the pipelines, none fulfilled the requirement of $\frac{2}{3}$ speedup efficiency when running on smaller datasets. All 15 steps eventually used a 10% downsampled dataset for performance profiling. Upon further investigation, we reasoned that the dynamic profiling approach works best if the tool utilizes multithreading and the number of data points are sufficiently large to leverage the increase of compute resources. Only a small fraction of the bioinformatics tools we tested utilized multithreading, and perhaps only BWA-MEM has fully developed this feature. Multithreading entails many benefits for scalable computing, and its optimized resource utilization could have a significant impact when large amounts of data processing are performed. Future improvement for bioinformatics tools' performances most likely include multithreading. We envision that with these improvements, the dynamic profiling feature in Hummingbird will be more broadly applicable and further reduce the cost of performance profiling.

Another important factor to consider when profiling bioinformatics tools and pipelines is Input/Output (I/O) throughput. In future versions of Hummingbird we plan to collect I/O performance metrics via the Hummingbird's profiler component and utilize it in the recommendation engine. This would enable Hummingbird to give the user recommendations about which disk to use along with their instance(standard hard disk drive (HDD) or solid-state drive (SSD).

Given the increasing demand of cloud computing and the need to perform analysis on ever larger study cohorts, we believe our performance prediction tool, Hummingbird, will contribute significantly to efficient bioinformatics computing in the cloud. The benefit derived from resource optimization should be particularly valuable for large consortia studies, such as the Trans-Omics for Precision Medicine (TOPMed) Taliun *et al.* (2019), the Centers for Common Disease Genomics (CCDG) Abel *et al.* (2020)and the Million Veteran Program (MVP) Gaziano *et al.* (2016).

4 Methods

4.1 Datasets

FASTQ files for NA12890 from the Illumina Platinum Genomes (Eberle *et al.*, 2017) served as the input for the GATK HaplotypeCaller pipeline. This dataset contains 707 million reads generated by Illumina HiSeq2000. All the FASTQ files are provided as open-source datasets on a public Google Cloud bucket (Illumina platinum genomes: `https://console.cloud.google.com/storage/browser/genomics-public-data/platinum-genomes`. Fastq files: `https://storage.cloud.google.com/genomics-public-data/references/GRCh37lite/GRCh37-lite.fa.gz`.

BAM files for a paired tumor and normal sample from pancreatic cancer, provided by the Texas Cancer Research Biobank (Becnel *et al.*, 2016), was used as the input for the GATK MuTect2 pipeline. The tumor BAM contains 86 million reads, and the normal BAM contains 81 million reads, both sequenced by Illumina HiSeq 2000 technology (Case #1 from the Texas Cancer Research Biobank:`http://txcrb.org/data.html`).

FASTQ files of an ATAC-seq experiment on primary keratinocytes from a male newborn from the ENCODE project (Davis *et al.*, 2018) (ATAC-seq experiment on primary keratinocytes from a male newborn: `https://www.encodeproject.org/experiments/ENCSR356KRQ/`) was used as the input for the ENCODE ATAC-seq pipeline; the two replicates were merged into one file, resulting in 308 million reads.

4.2 Downsampling FASTQ and BAM files

The tool seqtk was used (version 1.3 (r106) (`Seqtktool:https://github.com/lh3/seqtk`) to proportionally downsample FASTQ files to a defined fraction, such as 1%. For BAM files, we used the tool DownsampleSam from Picard (The picard command-line tools; `https://broadinstitute.github.io/picard/`) with the probability argument (P) to control the sampling fraction, e.g., P of 0.01 for retaining 1% of the reads in the BAM file. All downsampling used selection of random read-pairs. In Hummingbird's Memory Profiler step, systematic downsampling was set as 0.1%, 1% and 10%. In the Recommendation Engine step, a default downsampling ratio was set at 10%; in addition, when dynamic downsampling was enabled, performance profiling iterated from 0.1%, 1% to 10% of randomly downsampled data and stopped whenever the speedup efficiency met a preset threshold, which was set at 2/3 in our experiments. For each downsampled dataset, three runs were performed and the averaged value was used for performance analysis. In contrast, this downsampling step on input files in Hummingbird can be skipped if desired by the user. If the "fullrun" flag is set to true via modification of the user configuration file, Hummingbird bypasses the downsampling and proceeds to run on whole input files.

4.3 Pipelines

4.3.1 GATK HaplotypeCaller

Hummingbird was tested on the germline variant caller HaplotypeCaller in the Genome Analysis Toolkit (GATK 4) in command line format for calling germline SNVs (single nucleotide variants) and INDELs (insertions-deletions) (McKenna *et al.*, 2010). The docker image $broadinstitute/gatk$: 4.1.2.0 was used, and the best practices recommended by the GATK development team were followed (Van der Auwera *et al.*, 2013). The reference genome used was $GRCh37/hg19$.

4.3.2 GATK MuTect2

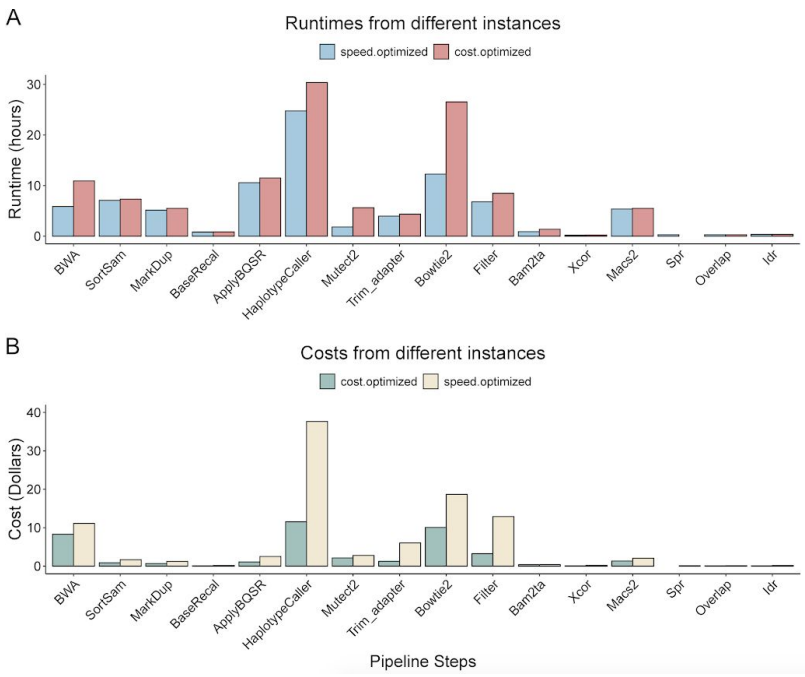
We investigated the somatic variant caller MuTect2 in the GATK 3.8 in command line format for calling somatic SNVs and INDELs (Cibulskis *et al.*, 2013). GATK 3.8 supported multithreading, while GATK 4 eliminated this option and instead supported distributed computing frameworks such as Apache Spark. Our testing of both GATK 4 and GATK 3.8 here captured different parallelization strategies.

4.3.3 ENCODE ATAC-seq

To demonstrate Hummingbird's ability to support bioinformatics pipelines presented in non-command line format, we tested on the ENCODE ATAC-seq pipeline v1.1.3 written in WDL and execution of the workflow jobs managed by Cromwell: `https://github.com/ENCODE-DCC/atac-seq-pipeline/tree/v1.1.3`.

4.4 Cloud Platforms

Our experiments were conducted on two major cloud computing platforms, GCP and AWS. In the context of cloud computing, both GCP and AWS use the term vCPU to represent the implementation of



C

Aggregated Runtime and Cost for Each Pipline

GATK HaplotypeCaller		
	Speed optimized	Cost optimized
Runtime (h)	54.29	66.47
Cost ($)	54.33	22.53

Mutect2		
	Speed optimized	Cost optimized
Runtime (h)	1.83	5.63
Cost ($)	2.79	2.14

ATAC-seq		
	Speed optimized	Cost optimized
Runtime (h)	30.44	47.19
Cost ($)	40.61	16.46

Fig. 3. Best instance types optimized for different purposes have a large impact on runtime and cost.(A) runtimes from speed-optimized compute instance (blue) and cost-optimized compute instance (red) for each workflow step. (B) Compute cost from cost-optimized instance (green) and cost-optimized instance (yellow) for each workflow step. (C) Runtime and costs for each pipeline, with all steps combined.

a single thread virtual CPU on an instance (vCPU concept on GCP: `https://cloud.google.com/compute/docs/faq`, vCPU concept on AWS: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-optimize-cpu.html`. We embraced this definition and implemented our system to consistently match the number of threads to the number of vCPUs if the software tool supports multithreading. When a bioinformatics software tool does not have a multithreading option, we typically choose two basic instances, 2 and 4 vCPUs instances, for example, to test performance.

4.4.1 Google Cloud Platform (GCP)

It offers a wide variety of compute instances, which can be largely categorized as high-CPU, standard and high-memory (high-mem) families. By design, Hummingbird can execute on all instance types to evaluate permutations of configurations. In practice, we selected instances with 8, 16 and 32 vCPUs in each category for showcasing the pipeline performances. All instance types on GCP used in our experiments are listed in Supplemental Table 1. For submitting jobs to a Google Compute instance, we utilized the command-line tool dsub v0.3.5 (Dsub, Google: `https://github.com/DataBiosphere/dsub`), in which the machine configuration, input and output paths, and the docker image can be specified for the execution.

4.4.2 Amazon Web Services (AWS)

Hummingbird uses AWS Batch as the backend to run pipelines on a cluster of instance types, as designated by the user. AWS also supports a wide variety of instances, and here we chose R4 and R5, two memory-optimized families, for the BWA-MEM experiment (Supplemental Table 2). Similar to that in GCP, Hummingbird handles the transmission of input and output files through storage buckets, in this case AWS S3. Other than the common configurations such as the machine configuration, input and output paths, etc., users will need to additionally supply a customized docker image with AWS CLI installed.

4.4.3 Microsoft Azure

Hummingbird uses Azure Batch as the backend to run pipelines on a pool of instance types, as designated by the user. Azure also supports a wide variety of instances, and here we chose Ev3 and Dv3 that are memory-optimized and standard families, respectively, for the BWA-MEM experiment (Supplemental Table 3). Similar to storage buckets in GCP, Hummingbird handles the transmission of input and output files through Azure Blob Storage in this case. Other than the common configurations such as the machine configuration, input and output paths, etc., users will need to additionally supply a customized docker image with Azure CLI installed.

4.5 Metrics for performance comparison between instances

4.5.1 Runtime & memory

Runtimes and memory usage were documented by the lightweight profiler GNU Time tool (/usr/bin/time). This tool was advantageous over the other tools that we tested, including valgrind (Valgrind: `https://valgrind.org/`), top (Top, task manager program in UNIX systems: `http://man7.org/linux/man-pages/man1/top.1.html`), free (Free tool: `http://man7.org/linux/man-pages/man1/free.1.html`), and cgroup (Cgroup, a Linux kernel feature: `http://man7.org/linux/man-pages/man7/cgroups.7.html`), in that it was easy to install in a Linux-like environment and compatible with the container environment in which our genomic applications run. In the case of genomic applications that can be run using a standalone program or script or installed software tool's command lines, runtimes were directly recorded. For genomic applications defined in WDL managed by Cromwell, in order to capture runtime, we used Cromwell's option to submit a script to a local backend running on a designated instance. The memory usage is recorded as the maximum resident set during the lifetime of a process in kilobytes. Essentially, it

Table 1. Metrics for comparing performance among instance types

Instance type	Num. of vCPUs	Exec. Time	Ideal Speedup	Speedup	Efficiency	Cost	Cost Efficiency
T_8(base)	P_8	T_8	$H_8{=}\frac{P_8}{P_8}$	$S_8{=}\frac{T_8}{T_8}$	$\frac{S_8}{H_8}$	C_8	$CE_8{=}\frac{S_8}{C_8}$
T_{16}	P_{16}	T_{16}	$H_{16}{=}\frac{P_{16}}{P_8}$	$S_{16}{=}\frac{T_8}{T_{16}}$	$\frac{S_{16}}{H_{16}}$	C_{16}	$CE_{16}{=}\frac{S_{16}}{C_{16}}$
T_{32}	P_{32}	T_{32}	$H_{32}{=}\frac{P_{32}}{P_8}$	$S_{32}{=}\frac{T_8}{T_{32}}$	$\frac{S_{32}}{H_{32}}$	C_{32}	$CE_{32}{=}\frac{S_{32}}{C_{32}}$

is the maximum heap size that the process has ever solely reached, and operating system or other services' memory usage is not included.

4.5.2 Cost

Runtimes were converted into costs according to the unit price of each cloud instance type, as publicly listed at the time these experiments were conducted (see Supplemental Tables 1-3 for details).

4.5.3 Speedup

Within each family of instance types, we set the instance with the least vCPU as the base instance. Then, we defined the speedup as the execution time run on the base instance over the execution time run on the investigated instance.

4.5.4 Speedup efficiency

We also introduced speedup efficiency to quantify the performance improvement in the unit processor, defined as the speedup over the fold change of vCPU. Ideally, when doubling the vCPU, the runtime will reduce to half, resulting in a speedup factor of 2 and a speedup efficiency of 1. However, in practice, speedup is often less than its full potential. For our experiments, we set the speedup threshold at $\frac{2}{3}$ to define a significant speedup efficiency.

4.5.5 Cost efficiency

To compare cost across all instances, we defined cost efficiency as the speedup over cost. This parameter helped to determine which instance type provided the most efficient speed increment in unit cost. The definitions of the performance metrics are summarized in Table 1.

Acknowledgments

We acknowledge the Stanford Genetics Bioinformatics Service Center (GBSC) for providing the gateway to GCP and AWS for this research. We thank members of the Cherry lab's ENCODE pipeline engineering team at Stanford University including J. Michael Cherry, J. Seth Strattan, Ulugbek K. Baymuradov, Jin-Wook Lee, Otto Jolanki and Casey Litton for feedback during the initial design and providing deep insights to the ENCODE pipeline features and architecture. We thank Keith Bettinger from the GBSC for his help in resolving infrastructure issues. We thank Derek Luan for his help in evaluating memory prediction methods and on improving the documentation during his GRIPS internship program at the GBSC during summer 2019. We thank members of the MVP bioinformatics team of Stanford University and VA Palo Alto for constructive feedback.

Funding

This work was supported by the Veterans Affairs Office of Research and Development Cooperative Studies Program, and by a grant from the National Human Genome Research Institute at the United States National Institutes of Health (U24 HG009397). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Human Genome Research Institute or the National Institutes of Health. This research also received support by the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Futures program. The funders had no role in design, data processing, implementation, decision to publish, or preparation of the manuscript.

References

Abel, H. J., Larson, D. E., Regier, A. A., Chiang, C., Das, I., Kanchi, K. L., Layer, R. M., Neale, B. M., Salerno, W. J., Reeves, C., *et al.* (2020). Mapping and characterization of structural variation in 17,795 human genomes. *Nature*, **583**(7814), 83–89.

Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., and Zhang, M. (2017). Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 469–482.

Becnel, L. B., Pereira, S., Drummond, J. A., Gingras, M.-C., Covington, K. R., Kovar, C. L., Doddapaneni, H. V., Hu, J., Muzny, D., McGuire, A. L., *et al.* (2016). An open access pilot freely sharing cancer genomic data from participants in texas. *Scientific data*, **3**(1), 1–10.

Cibulskis, K., Lawrence, M. S., Carter, S. L., Sivachenko, A., Jaffe, D., Sougnez, C., Gabriel, S., Meyerson, M., Lander, E. S., and Getz, G. (2013). Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature biotechnology*, **31**(3), 213–219.

Davis, C. A., Hitz, B. C., Sloan, C. A., Chan, E. T., Davidson, J. M., Gabdank, I., Hilton, J. A., Jain, K., Baymuradov, U. K., Narayanan, A. K., *et al.* (2018). The encyclopedia of dna elements (encode): data portal update. *Nucleic acids research*, **46**(D1), D794–D801.

Eberle, M. A., Fritzilas, E., Krusche, P., Källberg, M., Moore, B. L., Bekritsky, M. A., Iqbal, Z., Chuang, H.-Y., Humphray, S. J., Halpern, A. L., *et al.* (2017). 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*, **27**(1), 157–164.

Gaziano, J. M., Concato, J., Brophy, M., Fiore, L., Pyarajan, S., Breeling, J., Whitbourne, S., Deen, J., Shannon, C., Humphries, D., *et al.* (2016). Million veteran program: A mega-biobank to study genetic influences on health and disease. *Journal of clinical epidemiology*, **70**, 214–223.

Gunarathne, T., Wu, T.-L., Qiu, J., and Fox, G. (2010). Cloud computing paradigms for pleasingly parallel biomedical applications. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 460–469.

Hsu, C.-J., Nair, V., Freeh, V. W., and Menzies, T. (2017). Low-level augmented bayesian optimization for finding the best cloud vm. *arXiv preprint arXiv:1712.10081*.

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

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., *et al.* (2010). The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, **20**(9), 1297–1303.

O'Driscoll, A., Daugelaite, J., and Sleator, R. D. (2013). 'big data', hadoop and cloud computing in genomics. *Journal of biomedical informatics*, **46**(5), 774–781.

Stein, L. D. (2010). The case for cloud computing in genome informatics. *Genome biology*, **11**(5), 207.

Taliun, D., Harris, D. N., Kessler, M. D., Carlson, J., Szpiech, Z. A., Torres, R., Taliun, S. A. G., Corvelo, A., Gogarten, S. M., Kang, H. M., *et al.* (2019).

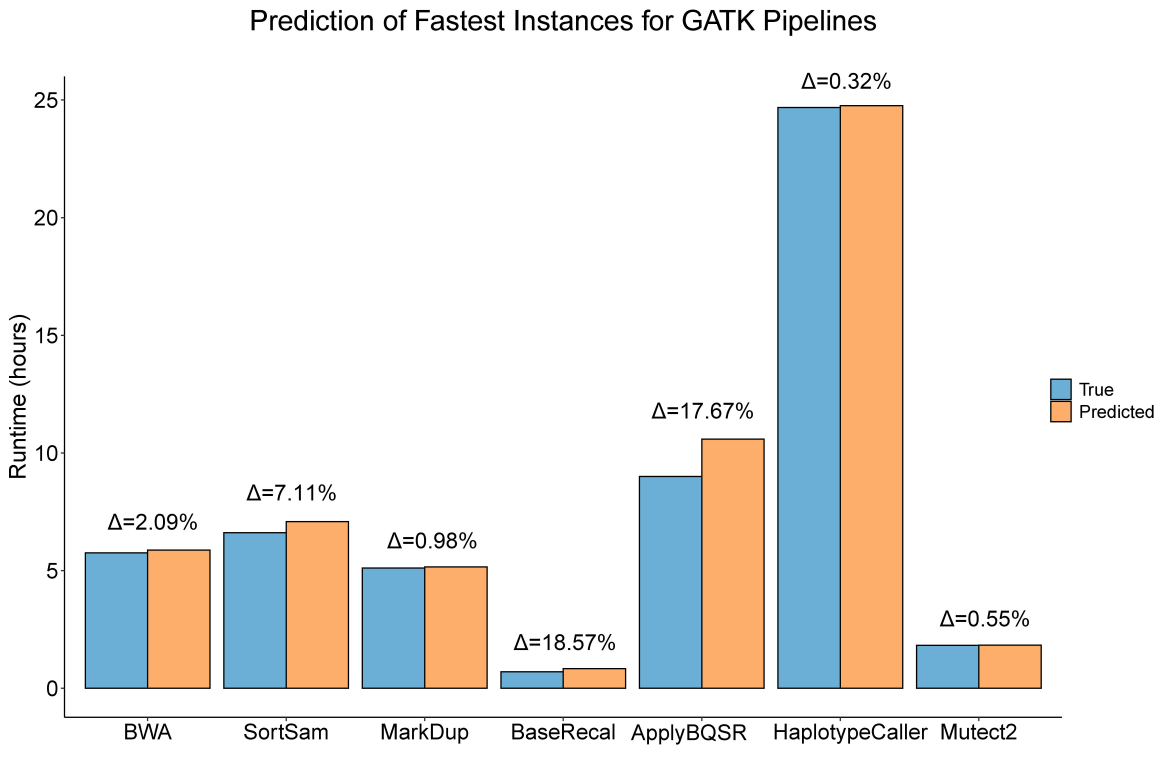
Sequencing of 53,831 diverse genomes from the nhlbi topmed program. *BioRxiv*, page 563866.

Van der Auwera, G. A., Carneiro, M. O., Hartl, C., Poplin, R., Del Angel, G., Levy-Moonshine, A., Jordan, T., Shakir, K., Roazen, D., Thibault, J., *et al.* (2013). From fastq data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. *Current protocols in bioinformatics*, **43**(1), 11–10.

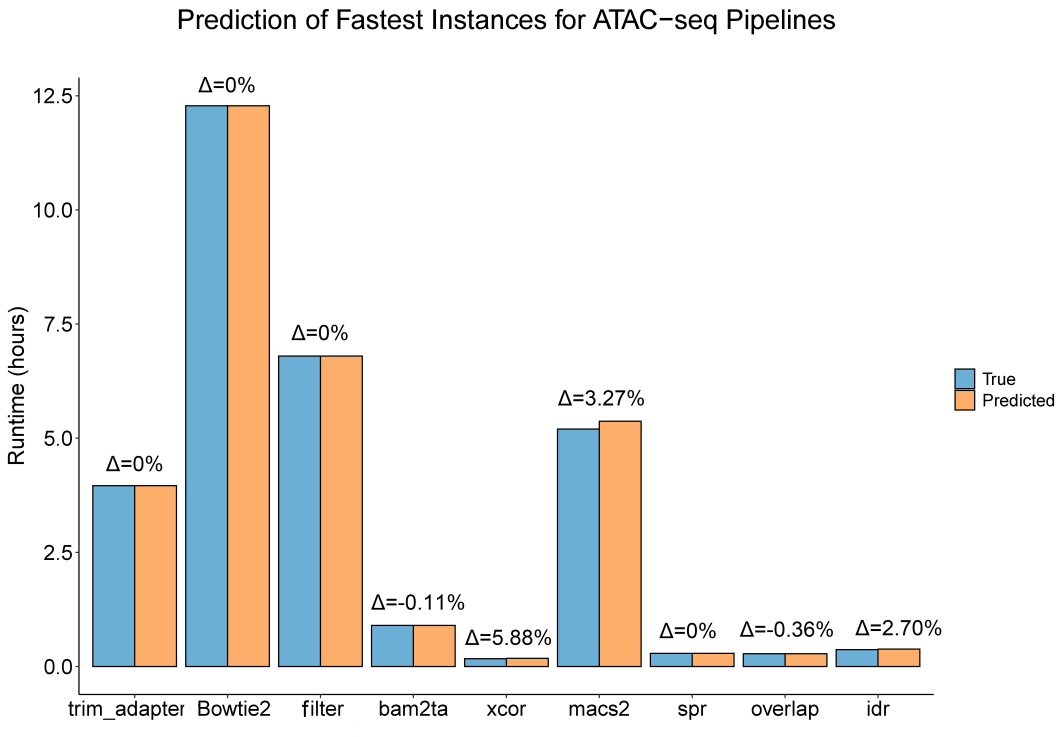
Venkataraman, S., Yang, Z., Franklin, M., Recht, B., and Stoica, I. (2016). Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 363–378.

Voss, K., Auwera, G. V. D., and Gentry, J. (2017). Full-stack genomics pipelining with gatk4+wdl+cromwell [version 1; not peer reviewed]. *ISCB Comm J*, **6**(1381).

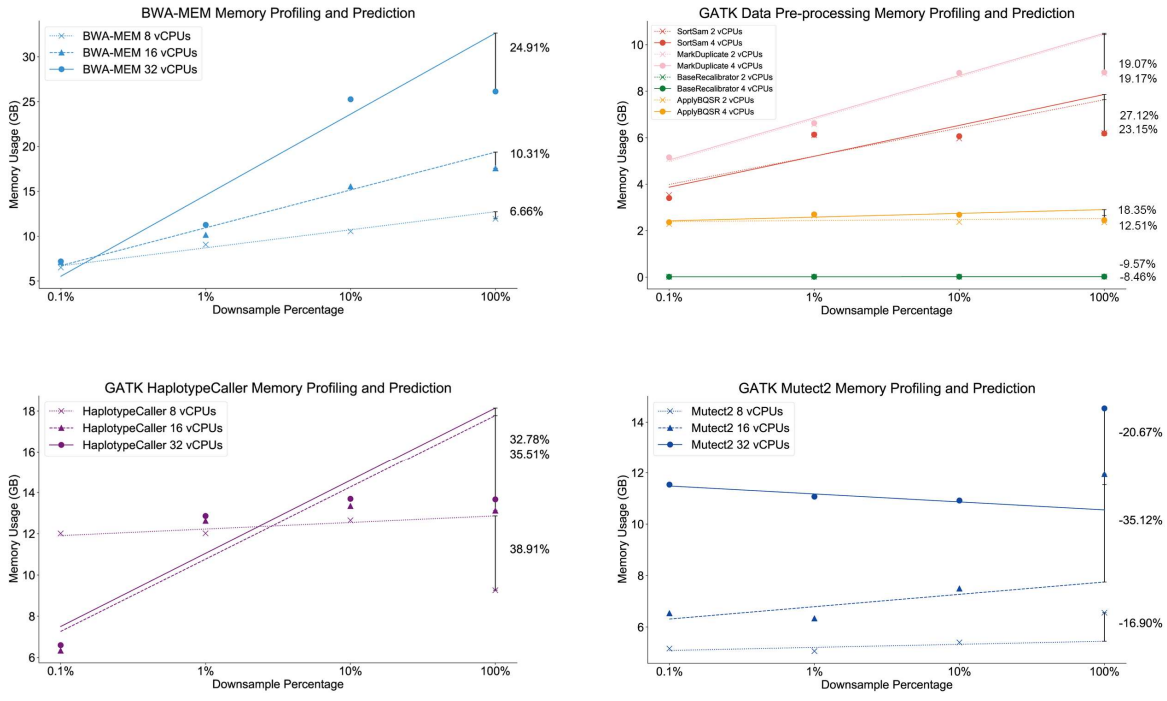
Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. (2017). Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 452–465.



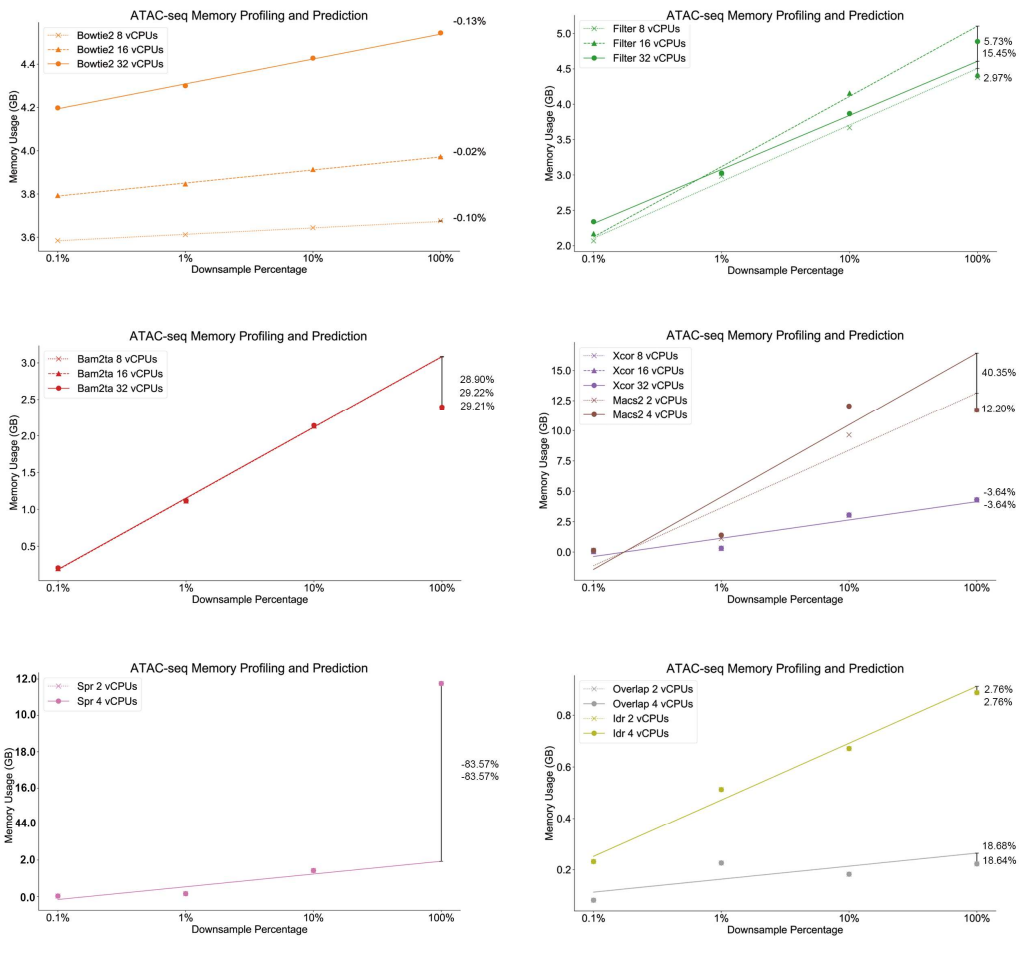
Supplemental Figure 1. Fastest compute instance for each step of the GATK pipelines: ground-truth vs Hummingbird prediction. The percentage differences in runtimes are indicated on top of each group.



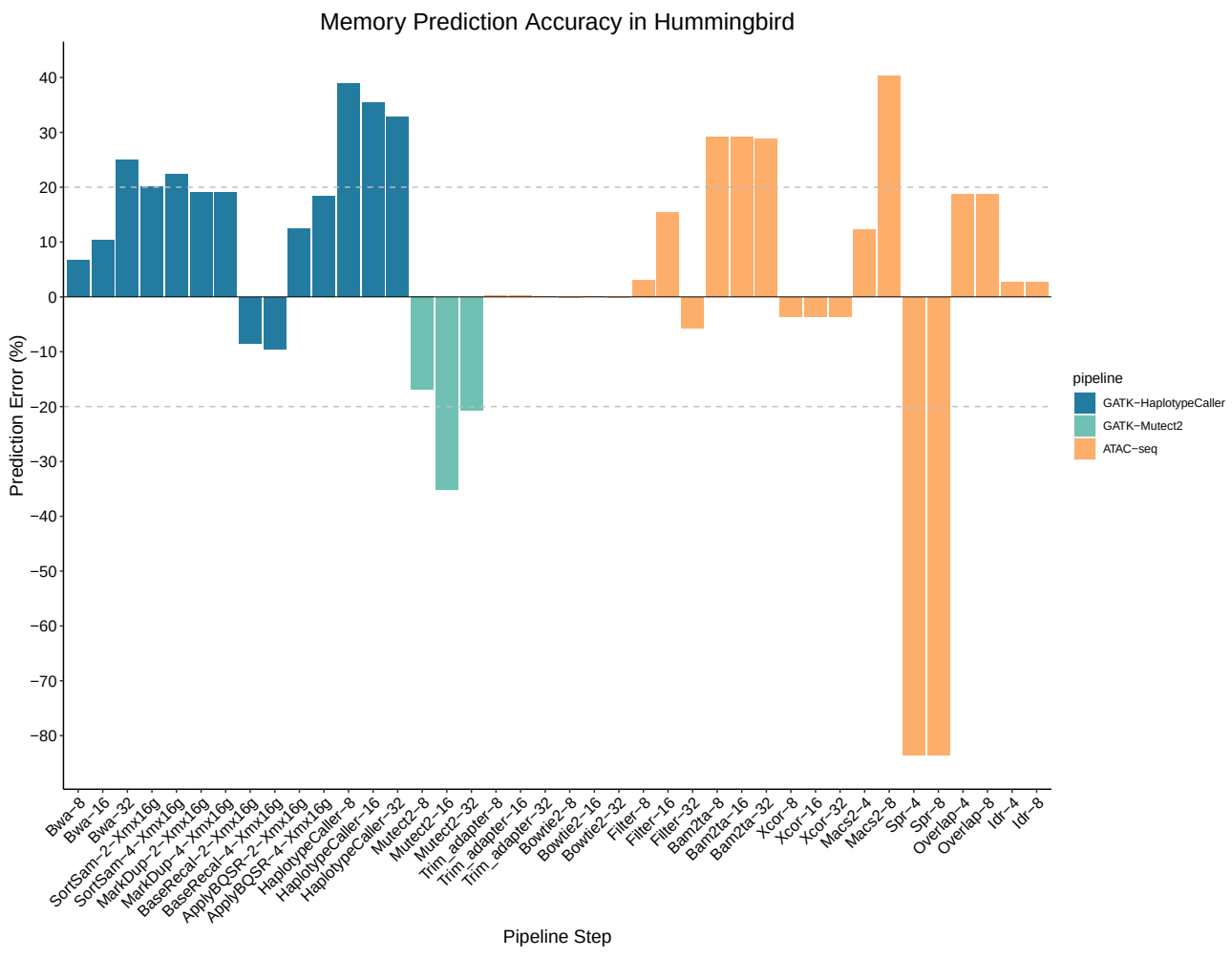
Supplemental Figure 2. Fastest compute instance for each step of the ATAC-seq pipeline: ground-truth vs Hummingbird prediction. The percentage differences in runtimes are indicated on top of each group.



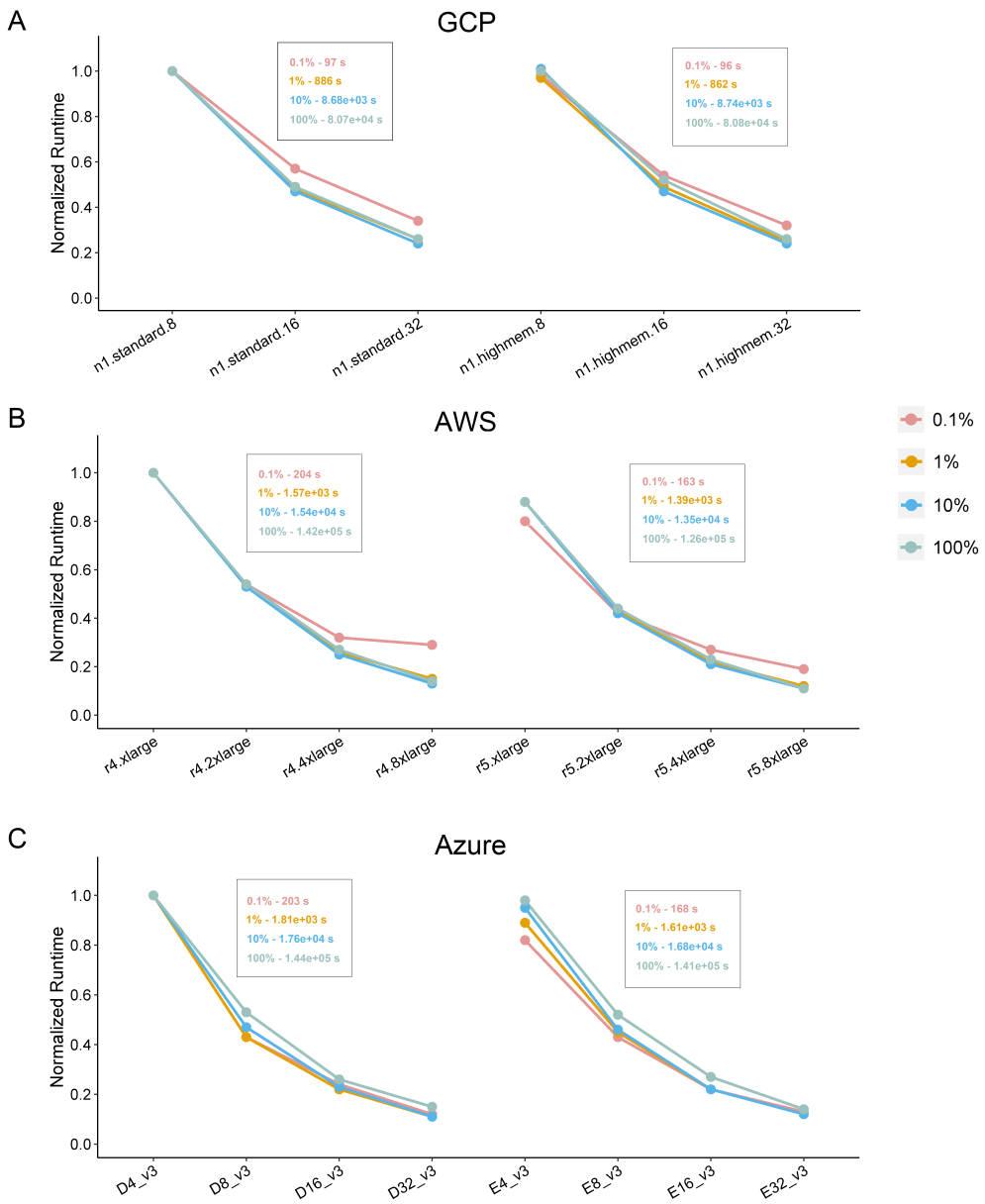
Supplemental Figure 3. Memory Prediction for GATK pipelines. Memories for running each pipeline step in the GCP n1-highmem family instance were recorded for datasets sampled at 0.1%, 1% and 10% of the whole input file. Linear Regression ($memory \sim \log_{10}(downsampling\ ratios)$) was applied to predict the memory for the whole input (100%). For comparison purposes, the actual memories for running the whole input were also recorded and are displayed. The predicted memory and the actual memory are connected by a line and the difference is indicated by the percentage value next to it. The instance types being tested by varying number of vCPUs are indicated by the legend boxes.



Supplemental Figure 4. Memory Prediction for the ATAC-seq pipeline. Similar to the memory prediction for the GATK pipelines, memories for running each step of the ATAC-seq pipeline in the GCP n1-highmem family instance were recorded for datasets sampled at 0.1%, 1% and 10% of the whole input file. Linear Regression ($memory \sim \log_{10}(downsampling\ ratios)$) was applied to predict the memory for the whole input (100%). For comparison purposes, the actual memories for running the whole input were also recorded and are displayed. The predicted memory and the actual memory are connected by a line and the difference is indicated by the percentage value next to it. The instance types being tested varying by number of vCPUs are indicated by the legend boxes.



Supplemental Figure 5. Memory prediction based on systematically downsampled datasets. The instance types being tested are denoted in the x-axis, with the numbers indicating the number of vCPUs in the instance type.



Supplemental Figure 6. Normalized runtimes of BWA-MEM on downsampled and whole input FASTQ data in various instance types on AWS cloud (A) and GCP cloud (B). The whole input dataset was NA12890 from the Illumina Platinum Genomes. In total, it contained 707 million reads, and the downsampled datasets were randomly sampled from the whole input in 0.1%, 1% and 10%. Two families of instances were considered for each cloud platform. Within each family of instances, compute resources (i.e., vCPU and memory) were doubled. Note that these instance types did not match exactly across the three cloud platforms. runtimes were normalized within each platform and dataset against the left-most instances in the figure (i.e., r4.xlarge for AWS, and n1.standard.8 for GCP).

Supplemental Table 1. GCP instance types tested in our experiments (pricing as of 2020-05-07 on us-west-2 region)

Instance Family	Instance Type	vCPUs	Memory(GB)	Price(hourly)
high-CPU	n1-highcpu-2	2	1.8	$0.0709
	n1-highcpu-4	4	3.6	$0.1418
	n1-highcpu-8	8	7.2	$0.28
	n1-highcpu-16	16	14.4	$0.57
	n1-highcpu-32	32	28.8	$1.13
Standard	n1-standard-2	2	7.5	$0.10
	n1-standard-4	4	15	$0.19
	n1-standard-8	8	30	$0.38
	n1-standard-16	16	60	$0.76
	n1-standard-32	32	120	$1.52
high-mem	n1-highmem-2	2	13	$0.12
	n1-highmem-4	4	26	$0.24
	n1-highmem-8	8	52	$0.47
	n1-highmem-16	16	104	$0.95
	n1-highmem-32	32	208	$1.89

Supplemental Table 2. AWS instance types tested in our experiments (pricing as of 2020-05-07 on US East (Ohio) Region)

Instance Family	Instance Type	vCPUs	Memory(GiB)	Price(hourly)
r4	r4.xlarge	4	30.5	$0.27
	r4.2xlarge	8	61	$0.53
	r4.4xlarge	16	122	$1.06
	r4.8xlarge	32	244	$2.13
r5	r5.xlarge	4	32	$0.25
	r5.2xlarge	8	64	$0.50
	r5.4xlarge	16	128	$1.01
	r5.8xlarge	32	256	$2.02

Supplemental Table 3. Azure instance types tested in our experiments (pricing as of 2020-12-20 on West US 2 Region)

Instance Family	Instance Type	vCPUs	Memory(GiB)	Price(hourly)
Dv3	D4 v3	4	16	$0.192
	D8 v3	8	32	$0.384
	D16 v3	16	64	$0.768
	D32 v3	32	128	$1.536
Ev3	E4 v3	4	32	$0.252
	E8 v3	8	64	$0.504
	E16 v3	16	128	$1.008
	E32 v3	32	256	$2.016

Supplemental Table 4. Runtime and cost for the memory profiling step for GATK pipelines in highmem instances. Sum cost: alignment: $11.19, intermediate steps: $2.83, HaplotypeCaller: $94.28, Mutect2: $9.54

	BWA	SortSam	MarkDup	BaseRecal	ApplyBQSR	HaplotypeCaller	Mutect2
runtime (min)	35.17	40.35	30.88	5.56	57.24	445.54	22.75
Cost ($)	$11.19	$0.83	$0.65	$0.11	$1.24	$94.28	$9.54

Supplemental Table 5. Runtime and cost for the memory profiling step for ATAC-seq pipelines in highmem instances. Total cost: $35.75

	trim_adaptor	Bowtie2	Filter	bam2ta	xcor	macs2	spr	overlap	idr
Runtime (min)	24.62	77.16	38.74	6.12	4.77	55.52	2.17	2.07	5.51
Cost ($)	$4.61	$18.19	$7.79	$1.26	$0.99	$2.37	$0.09	$0.09	$0.36

Supplemental Table 6. Performance profiling in all memory-matching instances for both the GATK pipelines. Total costs for the major pipeline steps were the following - alignment: $18.02, intermediate steps: $4.25, HaplotypeCaller: $189.89, and Mutect2: $8.86

	BWA	SortSam	MarkDup	BaseRecal	ApplyBQSR	HaplotypeCaller	Mutect2
# Instance types	6	3	3	4	4	7	8
Runtime (min)	145.7	45.97	37.62	5.87	72.19	468.55	23.89
Cost ($)	$18.02	$1.16	$0.90	$0.18	$2.01	$159.89	$8.86

Supplemental Table 7. Performance profiling in all memory-matching instances for ATAC-seq pipeline. Total cost ($): $70.89

	trim_adaptor	Bowtie2	Filter	bam2ta	xcor	macs2	spr	overlap	idr
# Instance types	9	8	8	9	8	3	5	5	5
Runtime (min)	28.14	171.26	56.34	9.18	6.61	56.83	2.2	2.12	7.64
Cost ($)	$9.48	$36.66	$16.34	$2.60	$1.93	$3.04	$0.17	$0.16	$0.51

Supplemental Table 8. AWS: Speedup and Cost Efficiency, to measure how effectively the tool accelerates given increased compute resources, are displayed for each instance type for the BWA-MEM runs. Within each cloud platform and for every downsampled dataset, the fastest instance with the least number of vCPUs was set as the baseline. All other instances were compared relative to the baseline by calculating the time speedup over vCPU fold changes. Our empirical experience suggests a speedup efficiency of or over $\frac{2}{3}$ as a strong positive indicator, implying the tool is able to effectively increase its performance when given more computational resources

.			Speedup			Cost Efficiency		
Instance Type	vCPU	Memory (GiB)	0.1%	1%	10%	0.1%	1%	10%
r4.xlarge	4	30.5	0.80	0.88	0.88	0.27	0.11	0.09
r4.2xlarge	8	61	1.50	1.64	1.64	0.48	0.19	0.15
r4.4xlarge	16	122	2.53	3.34	3.47	0.68	0.40	0.34
r4.8xlarge	32	244	2.77	5.74	6.72	0.41	0.59	0.64
r5.xlarge	4	32	1.00	1.00	1.00	0.45	0.15	0.12
r5.2xlarge	8	64	1.90	2.04	2.07	0.82	0.32	0.26
r5.4xlarge	16	128	2.95	4.00	4.13	0.98	0.61	0.51
r5.8xlarge	32	256	4.21	7.25	8.17	1.00	1.00	1.00

Supplemental Table 9. GCP version of Supplemental Table 8

			Speedup			Cost Efficiency		
Instance Type	vCPU	Memory (GiB)	0.1%	1%	10%	0.1%	1%	10%
n1-standard-8	8	30	0.98	0.97	1.00	0.46	0.27	0.24
n1-standard-16	16	60	1.73	2.01	2.15	0.72	0.57	0.55
n1-standard-32	32	120	2.89	3.76	4.09	1.00	1.00	1.00
n1-highmem-8	8	52	1.00	1.00	0.99	0.39	0.23	0.19
n1-highmem-16	16	104	1.82	2.00	2.11	0.63	0.45	0.43
n1-highmem-32	32	208	3.11	3.85	4.11	0.93	0.84	0.81

Supplemental Table 10. Azure version of Supplemental Table 8

.			Speedup			Cost Efficiency		
Instance Type	vCPU	Memory (GiB)	0.1%	1%	10%	0.1%	1%	10%
D4 v3	4	16	0.82	0.89	0.95	0.12	0.01	0.10
D8 v3	8	32	1.94	2.09	2.01	0.33	0.06	0.22
D16 v3	16	64	3.41	3.99	4.22	0.52	0.25	0.48
D32 v3	32	128	6.72	7.96	8.62	1.00	1.00	1.00
E4 v3	4	32	1.00	1.00	1.00	0.13	0.05	0.08
E8 v3	8	64	1.92	1.98	2.07	0.25	0.10	0.18
E16 v3	16	128	3.72	4.08	4.31	0.47	0.22	0.38
E32 v3	32	256	6.32	7.41	8.22	0.67	0.36	0.69

Supplemental Table 11. Instance types recommended by Hummingbird from three different cloud platforms to run BWA-MEM on NA12890 (707 million reads)

	Fastest Instance				Cheapest Instance			
	Instance type	vCPU	Memory	Runtime (h)	Instance type	vCPU	Memory	Cost
GCP	n1-highmem-32	32	208 GB	5.87	n1-standard-16	16	60 GB	$8.73
AWS	r5.8xlarge	32	256 GiB	4.46	r5.2xlarge	8	60 GiB	$8.73
Azure	Standard-E32-v3	32	256 GiB	5.49	Standard-E4-v3	4	32 GiB	$7.70