Leveraging Serverless Computing to Improve Performance for Sequence Comparison

Xingzhi Niu
School of Engineering and Technology
University of Washington
Tacoma, WA
nxz18@uw.edu

Dimitar Kumanov
School of Engineering and Technology
University of Washington
Tacoma, WA
dimitark@uw.edu

Ling-Hong Hung
School of Engineering and Technology
University of Washington
Tacoma, WA
lhhung@uw.edu

Wes Lloyd
School of Engineering and Technology
University of Washington
Tacoma, WA
wlloyd@uw.edu

Ka Yee Yeung
School of Engineering and Technology
University of Washington
Tacoma, WA
kayee@uw.edu

ABSTRACT

Cloud computing offers on-demand, scalable computing and storage, and has become an essential resource for the analyses of big biomedical data. The usual approach to cloud computing requires users to reserve and provision virtual servers. An emerging alternative is to have the provider allocate machine resources dynamically. This type of serverless computing has tremendous potential for biomedical research in terms of ease-of-use, instantaneous scalability, and cost effectiveness. In our proof of concept example, we demonstrate how serverless computing provides low cost access to hundreds of CPUs, on demand, with little or no setup. In particular, we illustrate that the all-against-all pairwise comparison among all unique human proteins can be accomplished in approximately 2 minutes, at a cost of less than $1, using Amazon Web Services Lambda. We also demonstrate the feasibility of our approach using Google Functions and show that the same task of pairwise protein sequence comparison can be accomplished in approximately 11.5 minutes. In contrast, running the same task on a typical laptop computer required 8.7 hours.

CCS CONCEPTS

• **Applied computing** → *Molecular sequence analysis*; • **Computer systems organization** → *Cloud computing*;

KEYWORDS

Cloud computing, bioinformatics, sequence alignment, serverless computing

ACM-BCB '19, September 7–10, 2019, Niagara Falls, NY, USA

ACM ISBN 978-1-4503-6666-3/19/09.
https://doi.org/10.1145/3307339.3343465

ACM Reference Format:
Xingzhi Niu, Dimitar Kumanov, Ling-Hong Hung, Wes Lloyd, and Ka Yee Yeung. 2019. Leveraging Serverless Computing to Improve Performance for Sequence Comparison. In *10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (ACM-BCB '19), September 7–10, 2019, Niagara Falls, NY, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3307339.3343465

1 INTRODUCTION

Cloud computing has become an essential resource in the analyses of big biomedical data by offering massive scalable computing and storage, data sharing, and on-demand access to resources and applications [14, 23]. Most major public cloud vendors offer tools for biomedical research, including Google Cloud Genomics [1], Amazon Web Services (AWS) [4] and Microsoft Genomics [9]. Currently, bioinformatics use cases benefit from leveraging cloud computing resources by provisioning on-demand computing infrastructure [10, 22, 31], providing secure access to protected data and scalable software platforms [13, 24].

A major barrier to widespread adoption of the cloud for large scale parallel bioinformatics jobs is the need to provision and configure virtual servers before computation can proceed. Setup requires expertise and can consume more time than the actual compute. Recently, serverless computing, through a cloud service delivery model referred to as Function-as-a-Service (FaaS), has emerged as an evolved simplified programming model that can be used to create scalable cloud applications with reduced configuration and management overhead [27]. Instead of the user reserving and provisioning a set of virtual servers, the cloud provider automatically allocates machine resources as needed. Specifically, serverless computing enables access to hundreds of CPUs on demand, with little or no setup. Code snippets can be easily deployed to the cloud in the form of microservices using serverless computing with resource provisioning, monitoring, scalability and fault tolerance provided automatically [12, 26].

In addition to the ease of use, these microservices can be launched quickly while requiring no provisioning or configuration of virtual machines and are inexpensive. Serverless computing enables a

true on-demand pricing model: there is no cost to the user when the user's function code is not being executed. This is in contrast to cloud service delivery models where the user is charged for cloud infrastructure even when virtual servers are idle. Most major public cloud vendors provide serverless FaaS computing capabilities, including AWS Lambda [3], Google Cloud Functions [6], Microsoft Azure Functions [8] and IBM Cloud Functions [7].

1.1 Related Work

Cole *et al.* [15] discussed best practices and tips to develop cloud computing solutions for biomedical informatics workflows. In particular, they proposed that serverless computing could ease the possible inconvenience for biology researchers to setup and configure the *IaaS* (infrastructure as a service) computing environment and this on-demand billing model could lower the cost of typical biological tasks in comparison with *IaaS*. However, serverless models may be a challenge to adopt for biomedical workflows, but there are many potential merits such as scalability [15].

The on-demand availability and high-concurrency offered by FaaS platforms makes them attractive for supporting biology research. Kijak *et al.* [20] discussed possible challenges for scheduling scientific workflows using FaaS platforms, including allocating resources to each task, estimating the execution time, and handling heterogeneity. They proposed a workflow scheduling algorithm adapted to serverless platforms and demonstrated feasibility using AWS Lambda.

In addition, many applications using serverless computing have been proposed in the literature. For example, Almeida *et al.* presented a novel tool [28] that used a serverless API to traverse an enormous number of medical records in a remote database. Lee *et al.* presented a tool [25] to visualize DNA sequences, which allows users to upload a DNA sequence in FASTA format to AWS S3 where the sequence is subsequently processed by AWS Lambda. In another example that combines serverless functions and cloud notifications, Hafeez *et al.* [19] proposed a method to utilize AWS Lambda and AWS Simple Queue Service (SQS) as a publisher/subscriber pattern to schedule serverless workflows. As another example, Gupta *et al.* provided a use case that performed approximate multiplication of matrices using AWS Lambda [18]. Their study showed that serverless computing offers an efficient on-demand solution to process large scale tasks at a lower cost.

1.2 Contributions

In this paper, we present a proof-of-concept case study performing sequence alignment comparing 20,000 protein sequences using serverless computing. Our case study demonstrates tremendous potential for leveraging serverless computing resources for biomedical research in terms of ease-of-use, instantaneous scalability, and cost effectiveness. While there are major advantages of serverless computing (i.e. users pay only for resources used when serverless functions are running), cloud providers place limits on the available computational resource for hosting serverless code including: the maximum number of concurrent client requests and the maximum memory and CPU resources available for each function invocation [12]. On AWS Lambda, functions execute with the following constraints: 500MB of disk space, 15 minutes maximum runtime, 3GB RAM, and 2 vCPUs. By default, Lambda functions are limited to 1,000 concurrent client requests per account, though this limit can be increased on request. We show that these constraints can be addressed using distributed approaches similar to those used in algorithms that leverage graphics processing units (GPUs). Here, we demonstrate the power and economy of scale of a serverless computing approach to simultaneously allocate hundreds of CPUs to perform pairwise protein sequence alignment, a computationally intensive bioinformatics task, in significantly less execution time using both AWS Lambda and Google Cloud Functions.

2 PAIRWISE PROTEIN SEQUENCE COMPARISON

Smith-Waterman [29] is a dynamic programming algorithm for comparing protein sequences. Performing similarity comparisons of protein sequences is a common precursor to clustering sequences into related families or functional groups. The Similarity Matrix of Proteins (SIMAP) project applies Smith-Waterman to compare all-against-all pairs of known protein sequences covering all major protein public databases, a task that is expected to take years to compute [11]. For our demonstration, we perform the all-against-all pairwise comparison with human proteins using the Striped Smith-Waterman (SSW) implementation [32].

Our input data consist of 20,336 unique human protein sequences from a freely accessible database of protein sequence and functional information called UniProt [16]. We partition these 20,336 sequences into 41 subsets, each consisting of approximately 500 sequences. The calculation of all the pairwise interactions between sequences in a pair of subsets (i.e. 500 vs. 500 proteins) is small enough to be run by individual functions on a serverless cloud platform with 1.5 GB memory capacity. This results in a total of 861 unique tasks to compare each of these 41 subsets to another subset including itself.

3 EXPERIMENTAL DESIGN

3.1 Method

3.1.1 AWS Lambda. To run a comparison task using AWS Lambda, the user bundles a script with the required dependencies and uploads them. The function executes on Lambda using Amazon Linux [2]. The 41 subsets of protein sequences are stored in Amazon's Simple Storage Service (S3).

To profile the performance of running Smith-Waterman on AWS Lambda, we deployed a scheduler script written in Python on an Elastic Compute Cloud (EC2) instance, and on a personal laptop to coordinate the execution of all 861 tasks concurrently. Use of cloud-based virtual machines is not required, since AWS Lambda functions (and Google Cloud Functions) can be invoked by any client. This script acts as the client to invoke AWS Lambda independently for each respective task utilizing the Striped Smith-Waterman (SSW) executable. Our Lambda function writes similarity scores to a file on S3, and reports completion by sending a message to the AWS Simple Queue Service (SQS). The client Python script monitors SQS for messages to determine when the whole job has completed all 861 tasks. Our code and documentation are publicly available on GitHub. (https://github.com/BioDepot/TaskPerform_AWSLambda). An overview of our approach is summarized in Figure 1.

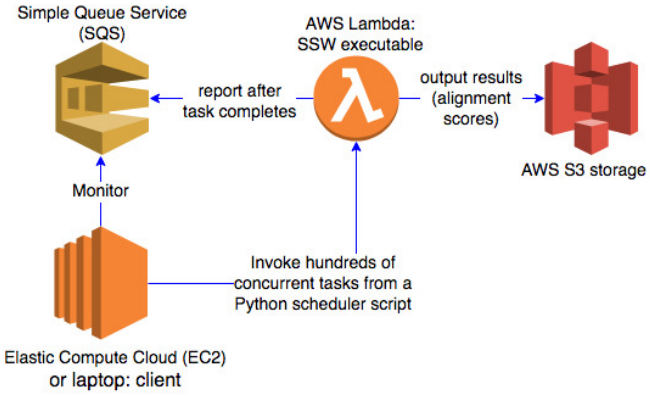

Figure 1: Overview of our approach using AWS Lambda. The Python scheduler script running on an Elastic Compute Cloud (EC2) instance or a personal laptop invokes hundreds of concurrent tasks on AWS lambda in parallel. In our case study, a Python scheduler script reads two subsets of proteins (each subset consists of approximately 500 protein sequences), calls the Striped Smith-Waterman (SSW) executable, writes the similarity scores to a file on S3, and reports completion by sending a message to the AWS Simple Queue Service (SQS), a message queuing service. The EC2 instance or the laptop monitors SQS to determine when the whole job has completed.

3.1.2 Google Cloud Functions. A comparison implementation having the same dependencies and script was produced to evaluate the performance of performing sequence alignment using Google Cloud Functions. The interface is modified slightly to leverage Google Pub/Sub in place of Amazon SQS. We deployed a client to invoke Google Cloud Functions using a Google Compute Engine virtual machine [5]. The 41 subsets of protein sequences are stored in Google Cloud Storage.

To benchmark the performance of Smith-Waterman on Google Cloud Functions, we modified the scheduler script and deployed it on a Google Compute Engine instance, and on a personal laptop, to orchestrate the execution of all 861 tasks concurrently. This script acts as the client to invoke Google Cloud Functions independently for each respective task utilizing the Striped Smith-Waterman (SSW) executable. Google Cloud Functions writes similarity scores to a file on Google Storage, and reports completion by sending a message to the Google Pub/Sub. The client Python script subscribes and pulls Pub/Sub messages to determine when the whole job has completed all 861 tasks. An overview of our approach is summarized in Figure 2. Our code and documentation for Google Cloud are publicly available on GitHub. (https://github.com/BioDepot/smith-waterman-serverless).

3.2 Experimental Setup

We compared the execution time of invoking the AWS Lambda functions using an EC2 client versus a laptop computer. In addition, we also benchmarked the runtime of performing sequence comparison directly on a laptop computer without leveraging cloud resources. We also compared the execution time of invoking Google

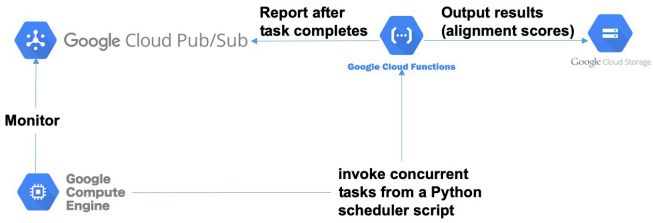

Figure 2: Overview of our approach using Google Cloud Functions. The Python scheduler script running on a Google Compute Engine instance or a personal laptop invokes hundreds of concurrent tasks on Google Cloud Functions in parallel. The similarity scores computed by the Google Cloud Functions are sent to a file on Google Cloud Storage. We use the Google Cloud Pub/Sub, a real-time messaging service, to report completion of tasks executed on the Google Cloud Functions.

Cloud Functions using a client deployed on Google Compute Engine versus a laptop computer.

3.2.1 EC2 client and AWS Lambda. We repeated our empirical experiments invoking AWS Lambda functions with 1.5GB memory capacity from an EC2 VM client (m5.2xlarge that consists of 8 virtual central processing units and 32G of memory) ten times. The observed execution time to complete all 861 Lambda function calls ranged from 73 seconds to 87 seconds, with a mean of 77 seconds (=1.28 minutes) and a median of 74 seconds (=1.25 minutes).

3.2.2 Laptop client and AWS Lambda. We repeated our empirical experiments three times where we invoked AWS Lambda from our laptop client (Intel i5-7200U CPU 2-core processor @ 2.50 GHz, 8 GB of RAM). The observed execution time to complete all 861 concurrent Lambda function calls ranged from 128 seconds to 140 seconds, with a mean of 132 seconds (=2.20 minutes), and a median of 129 seconds (=2.15 minutes).

3.2.3 Laptop without AWS Lambda. Our empirical evaluation of sequence alignment performance on a laptop leveraged the same computer with an Intel i5-7200U 2-core CPU running Ubuntu 16.04 on Oracle VM Virtual Box with the following settings: 4096 MB RAM, 1 processor core at 100% execution cap.

3.2.4 Google Compute Engine and Cloud Functions. We repeated our empirical experiments ten times invoking Google Cloud Functions with 1GB memory capacity from a Google Compute Engine with similar specifications as the VM used in the AWS experiments. Specifically, we used a n1-standard-8 instance consisting of 8 vCPUs and 30G of memory. The observed execution time completing all 861 concurrent Lambda functions ranges from 579 seconds to 695 seconds, with a mean of 666 seconds (=11.10 minutes), and a median of 674 seconds (=11.23 minutes).

3.2.5 Laptop client and Cloud Functions. We repeated our empirical experiments invoking Google Cloud Functions with 1.5GB memory capacity from our laptop client (AMD 6376 16-core processor @ 2.30 GHz, 32 GB of RAM) 10 times. The observed execution

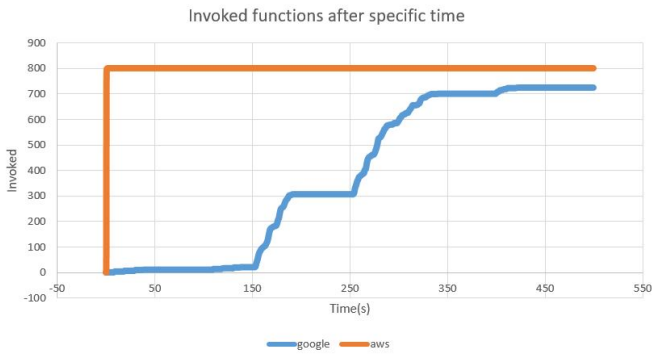

Figure 3: Number of AWS Lambda and Google Cloud Functions invocations after specific time period. We observe that 800 AWS Lambda functions can be invoked in a couple of seconds, while it takes more than 5 minutes to invoke 700 Google Cloud Functions.

time completing all 861 concurrent Lambda functions ranges from 647 seconds to 799 seconds, with a mean of 709 seconds (=11.82 minutes), and a median of 686 seconds (=11.43 minutes).

4 EXPERIMENTAL RESULTS

4.1 AWS Lambda

When the Lambda functions are invoked by the Python script running on a laptop computer, the average execution time is 2.20 minutes. This is in contrast to a run time of 8.7 hours when performing sequence alignment on the laptop without using AWS Lambda. Similarly, the average execution time is just 1.28 minutes when the Lambda functions are invoked by the Python script running on an AWS EC2 instance. The estimated hosting cost for sequence alignment using AWS Lambda for either client is under $1. Please refer to the Methods section for details of our empirical experiments. Table 2 shows details of our cost analyses.

4.2 Google Cloud Functions

When the Google Cloud Functions are invoked by the Python script running on a laptop computer, the average execution time is 11.82 minutes. This is much faster than a run time of 8.7 hours when performing all the tasks on the laptop without using cloud functions, while it is 6 times slower than that running on AWS Lambda. The main reason appears to be the scaling performance for invoking many concurrent calls to Google Cloud Functions. Google Cloud Functions requires a considerably longer time to scale up and complete 861 function invocations. Figure 3 shows the relation between launch time and number of invoked functions. Our results show that 800 AWS Lambda functions can be invoked in a couple of seconds. In contrast, it takes more than 5 minutes to invoke 700 Google Cloud Functions and more than 7 minutes to invoke 800 Google Cloud Functions. As a result of the invocation rate, the average execution time is 11.1 minutes when the Google Cloud Functions are invoked by the Python script running on an Google Compute Engine VM.

4.3 Discussion

Table 1 and Table 2 summarize the observed execution time and hosting costs. We observe significant improvement in execution time using either AWS Lambda or Google Cloud Functions. Specifically, the total execution time averaged over 10 repeated experiments is 77 seconds using an AWS EC2 client and Lambda: a 407x speedup compared to using a laptop computer. The total execution time averaged over 10 repeated experiments is 666 seconds using a Google Compute Engine client and Cloud Functions: a 47x speedup compared to using a laptop computer. The longer execution time for Google Cloud Functions is primarily due to the fact that the invocation time of hundreds of Cloud Functions is longer than AWS Lambda. In terms of cost, we can accomplish this task of pairwise protein sequence comparison on both AWS Lambda and Google Cloud Functions for under $1.

5 CONCLUSIONS

In this paper, we demonstrate how a computationally intensive bioinformatics task can leverage serverless cloud computing to dramatically speed up execution time at low cost. Our divide-and-conquer strategy can be applied to other problems that can be partitioned into small sub-problems. This approach is commonly leveraged for example, in GPU implementations, where a large task is divided into sub-tasks that are operated on by individual GPU processor cores. Sequence alignment, protein-folding, and deep-learning are computationally intensive bioinformatics tasks with GPU implementations [17, 21, 30] where a serverless computing approach may prove useful. With the ability of serverless cloud computing to quickly leverage hundreds of CPUs, computational power that was once the exclusive domain of supercomputers is now easy to access, available on demand, and at low cost, to help solve resource intensive bioinformatics problems.

ACKNOWLEDGMENTS

This research is supported by the NIH grant R01 GM126019, and the US National Science Foundation's Advanced Cyberinfrastructure Research Program (OAC-1849970). We would also like to acknowledge support from the AWS Cloud Credits for Research (to Hung, Lloyd and Yeung) and Google Cloud (to Lloyd and Yeung) for providing cloud computing resources.

REFERENCES

[1] Google Genomics. https://cloud.google.com/genomics/
[2] Amazon Linux AMI. https://aws.amazon.com/amazon-linux-ami
[3] AWS Lambda. https://aws.amazon.com/lambda/
[4] Biotech and Pharma in the Cloud: accelerate scientific discovery and enable operational efficiencies in the AWS cloud. https://aws.amazon.com/health/biotech-pharma/
[5] Google Cloud Compute Engine. https://cloud.google.com/compute/
[6] Google Cloud Functions. https://cloud.google.com/functions/
[7] IBM Cloud Functions. http://www.ibm.com/cloud/functions
[8] Microsoft Azure Functions: Build apps faster with a serverless architecture. https://azure.microsoft.com/en-us/services/functions/
[9] Microsoft Genomics. https://azure.microsoft.com/en-us/services/genomics/
[10] Enis Afgan, Dannon Baker, Nate Coraor, Hiroki Goto, Ian M Paul, Kateryna D Makova, Anton Nekrutenko, and James Taylor. 2011. Harnessing cloud computing with Galaxy Cloud. *Nature biotechnology* 29, 11 (2011), 972.
[11] Roland Arnold, Florian Goldenberg, Hans-Werner Mewes, and Thomas Rattei. 2013. SIMAPâĂŤthe database of all-against-all protein sequence similarities and annotations with new interfaces and increased coverage. *Nucleic acids research* 42, D1 (2013), D279–D284.

	AWS EC2 client and Lambda	Google Compute Engine and Functions	Laptop (no serverless)
Average Total Time (seconds)	77	666	31320
Function Average CPU Time (seconds)	76	67	Not Applicable
Function Average Launch Time (seconds)	1	224	Not Applicable

Table 1: Comparison of total time, average execution (CPU) time and average launch time. The total execution time averaged over 10 repeated experiments is 77 seconds using an AWS EC2 client and Lambda: a 407x speedup compared to using a laptop computer. The total execution time averaged over 10 repeated experiments is 666 seconds using a Google Compute Engine client and Function: a 47x speedup compared to using a laptop computer. Use of a cloud-based virtual machine as a client is not required, since Lambda functions and Cloud Functions can be invoked by any client. The launch time of Google Cloud Functions is longer than AWS Lambda, while the execution time is shorter as Google Cloud Functions use RAM for local temporary storage instead of disk of AWS Lambda.

	AWS EC2 client and Lambda	Google Compute Engine and Functions
Cloud Function Costs	$0.83306	$0.73590
Notification Queue Costs	$0.00103	$0.00000
Instance Costs	$0.03413	$0.03455
Storage Costs	$0.01994	$0.02220
Total Costs	**$0.88817**	**$0.79265**

Table 2: Cost breakdown of AWS workflow with Lambda and EC2 and Google workflow with Cloud Functions and Compute Engine. In our analyses, we accounted for the fact that Google Pub/Sub has 10 GB free tier and 50GB for cloud storage each month.

[12] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*. Springer, 1–20.

[13] Chet Birger, Megan Hanna, Edward Salinas, Jason Neff, Gordon Saksena, Dimitri Livitz, Daniel Rosebrock, Chip Stewart, Ignaty Leshchiner, Alexander Baumann, et al. 2017. FireCloud, a scalable cloud-based platform for collaborative genome analysis: Strategies for reducing and controlling costs. *bioRxiv* (2017), 209494.

[14] Barbara Calabrese and Mario Cannataro. 2016. *Cloud Computing in Bioinformatics: current solutions and challenges.* Technical Report. PeerJ Preprints.

[15] Brian S. Cole and Jason H. Moore. 2018. Eleven quick tips for architecting biomedical informatics workflows with cloud computing. *PLOS Computational Biology* 14, 3 (03 2018), 1–11. https://doi.org/10.1371/journal.pcbi.1005994

[16] UniProt Consortium et al. 2018. UniProt: the universal protein knowledgebase. *Nucleic acids research* 46, 5 (2018), 2699.

[17] Andreas W Gotz, Mark J Williamson, Dong Xu, Duncan Poole, Scott Le Grand, and Ross C Walker. 2012. Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized born. *Journal of chemical theory and computation* 8, 5 (2012), 1542–1555.

[18] Vipul Gupta, Shusen Wang, Thomas Courtade, and Kannan Ramchandran. 2018. OverSketch: Approximate Matrix Multiplication for the Cloud. 298–304. https://doi.org/10.1109/BigData.2018.8622139

[19] Faisal Hafeez, Pezhman Nasirifard, and Hans-Arno Jacobsen. 2018. A Serverless Approach to Publish/Subscribe Systems. In *Proceedings of the 19th International Middleware Conference (Posters) (Middleware '18)*. ACM, New York, NY, USA, 9–10. https://doi.org/10.1145/3284014.3284019

[20] J. Kijak, P. Martyna, M. Pawlik, B. Balis, and M. Malawski. 2018. Challenges for Scheduling Scientific Workflows on Cloud Functions. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 460–467. https://doi.org/10.1109/CLOUD.2018.00065

[21] Petr Klus, Simon Lam, Dag Lyberg, Ming Sin Cheung, Graham Pullan, Ian McFarlane, Giles SH Yeo, and Brian YH Lam. 2012. BarraCUDA-a fast short read sequence aligner using graphics processing units. *BMC research notes* 5, 1 (2012), 27.

[22] Alexander Lachmann, Denis Torre, Alexandra B Keenan, Kathleen M Jagodnik, Hoyjin J Lee, Lily Wang, Moshe C Silverstein, and Avi MaâĂŹayan. 2018. Massive mining of publicly available RNA-seq data from human and mouse. *Nature communications* 9, 1 (2018), 1366.

[23] Ben Langmead and Abhinav Nellore. 2018. Cloud computing for genomic data analysis and collaboration. *Nature Reviews Genetics* 19, 4 (2018), 208.

[24] Jessica W Lau, Erik Lehnert, Anurag Sethi, Raunaq Malhotra, Gaurav Kaushik, Zeynep Onder, Nick Groves-Kirkby, Aleksandar Mihajlovic, Jack DiGiovanna, Mladen Srdic, et al. 2017. The Cancer Genomics Cloud: collaborative, reproducible, and democratizedâĂŤa new paradigm in large-scale computational research. *Cancer research* 77, 21 (2017), e3–e6.

[25] Benjamin D Lee, Michael A Timony, and Pablo Ruiz. 2019. DNAvisualization.org: a serverless web tool for DNA sequence visualization. *Nucleic Acids Research* (06 2019). https://doi.org/10.1093/nar/gkz404 arXiv:http://oup.prod.sis.lan/nar/advance-article-pdf/doi/10.1093/nar/gkz404/28772452/gkz404.pdf

[26] Wes Lloyd, Shruti Ramesh, Swetha Chinthalapati, Lan Ly, and Shrideep Pallickara. 2018. Serverless computing: An investigation of factors influencing microservice performance. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 159–169.

[27] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. 2017. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 162–169.

[28] Jonas S. Almeida, Janos Hajagos, Joel Saltz, and Mary Saltz. 2019. Serverless OpenHealth at data commons scale âĂŤ Traversing the 20 million patient records of New YorkâĂŹs SPARCS dataset in real-time. *PeerJ* 7 (01 2019), e6230. https://doi.org/10.7717/peerj.6230

[29] Temple F Smith, Michael S Waterman, et al. 1981. Identification of common molecular subsequences. *Journal of molecular biology* 147, 1 (1981), 195–197.

[30] The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688* (2016).

[31] John Vivian, Arjun Arkal Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D Deran, Audrey Musselman-Brown, et al. 2017. Toil enables reproducible, open source, big biomedical data analyses. *Nature biotechnology* 35, 4 (2017), 314.

[32] Mengyao Zhao, Wan-Ping Lee, Erik P Garrison, and Gabor T Marth. 2013. SSW library: an SIMD Smith-Waterman C/C++ library for use in genomic applications. *PloS one* 8, 12 (2013), e82138.