Briefings in Bioinformatics, 00(0), 2021, 1–11

https://doi.org/10.1093/bib/bbab349
Review

Serverless computing in omics data analysis and integration

Piotr Grzesik, Dariusz R. Augustyn, Łukasz Wyciślik and Dariusz Mrozek

Corresponding author: Dariusz Mrozek, Department of Applied Informatics, Silesian University of Technology, Gliwice 44-100, Poland.
E-mail: Dariusz.Mrozek@polsl.pl

Abstract

A comprehensive analysis of omics data can require vast computational resources and access to varied data sources that must be integrated into complex, multi-step analysis pipelines. Execution of many such analyses can be accelerated by applying the cloud computing paradigm, which provides scalable resources for storing data of different types and parallelizing data analysis computations. Moreover, these resources can be reused for different multi-omics analysis scenarios. Traditionally, developers are required to manage a cloud platform's underlying infrastructure, configuration, maintenance and capacity planning. The serverless computing paradigm simplifies these operations by automatically allocating and maintaining both servers and virtual machines, as required for analysis tasks. This paradigm offers highly parallel execution and high scalability without manual management of the underlying infrastructure, freeing developers to focus on operational logic. This paper reviews serverless solutions in bioinformatics and evaluates their usage in omics data analysis and integration. We start by reviewing the application of the cloud computing model to a multi-omics data analysis and exposing some shortcomings of the early approaches. We then introduce the serverless computing paradigm and show its applicability for performing an integrative analysis of multiple omics data sources in the context of the COVID-19 pandemic.

Key words: cloud computing; serverless computing; omics data processing; omics data integration; function-as-a-service; container-as-a-service; bioinformatics

Introduction

In recent years, data produced by genomic, transcriptomic, proteomic and metabolomic experiments have been fed into an increasing variety of data sources and multi-omics data sets. These data sets provide an important source of knowledge for studying serious diseases such as Alzheimer's disease, Parkinson's disease, cancer or coronavirus disease (COVID-19). The complex, multi-step analysis pipelines that are used to find potential drugs to cure these diseases usually require appropriate data integration that is driven by time-consuming

Piotr Grzesik, M.Sc., Eng., is a PhD student at the Department of Applied Informatics at the Silesian University of Technology in Gliwice. His main research areas include cloud computing, serverless computing, the Internet of things, edge computing, databases and bioinformatics.
Dariusz R. Augustyn, Ph.D., Eng., works in the Department of Applied Informatics at the Silesian University of Technology in Gliwice. His main research areas include distributed and parallel processing, cloud computing, database theory, mathematical modeling of dynamical systems and software engineering.
Łukasz Wyciślik, Ph.D., Eng., has been affiliated with the Department of Applied Informatics at the Silesian University of Technology for over 20 years. His scientific interests focus on data processing, databases, systems modeling, software engineering and artificial intelligence. As an employee of the Department of Applied Informatics, he places great emphasis on the possibility of practical applications of his research results.
Dariusz Mrozek, Ph.D., D.Sc., Eng., is currently an associate professor, Head of Department of Applied Informatics and Deputy Dean for Cooperation and Development at the Faculty of Automatic Control, Electronics and Computer Science at the Silesian University of Technology in Gliwice, Poland. His research interests cover the Internet of things, information systems, artificial intelligence, parallel and cloud computing, databases and big data and bioinformatics. He is currently focused on applying AI-supported IoT technologies in various areas, from manufacturing to life sciences.
Submitted: 27 July 2021; **Received (in revised form):** 28 June 2021

computations. For example, in response to the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) pandemic, the scientific community has sought to create a shortlist of drugs that can fight the infection rapidly, by repurposing existing drugs. Such approaches rely on identifying differentially expressed genes via transcriptomic analyses and metabolites via proteomic and metabolomic experiments. In addition, differentially abundant proteins must be identified, along with protein–protein interactions between the studied pathogen and the host organism [50]. This is usually followed by a genome-wide association study, further transcriptomic analysis and investigation of pathogen–host interaction networks. The resultant ranked drug lists are then re-ranked based on integrated gene interaction networks that have undergone functional analysis and scoring, structural clustering or proteome profiling [36]. The majority of the data analysis underlying this process requires computational power, resources and storage for various types of omics data. The source of the computational power depends on the analysis in question. These sources can be classified as follows:

1. Local workstations
2. Local data centers
3. Cloud computing

Local workstations (for example, desktop computers) are the simplest approach and the most common platform for the calculations required for an omics data analysis. Analysts typically have convenient access to a workstation and have a high degree of control over configuration and process management. The desktop versions of tools, such as Taverna [22, 42] or Galaxy server [28], allow for the construction of complex analysis workflows and integration of different software components and data sources. However, executing the workflows on local workstations has limitations. Even as central processing unit (CPU) performance and core number increases, standard workstations used for desktop computations still have very limited computing capabilities. Moreover, although the cost of storage space per gigabyte is decreasing, and the capacities of hard disk drives and solid state drives are increasing, desktop computers are still limited in terms of storage. This renders desktop computing insufficient to deal with the large quantity of biological data that is produced, for example, by next-generation sequencing experiments or 3rd-generation sequencing techniques, such as Oxford MinION Nanopore.

Large laboratories, hospitals, universities and other institutions that perform a biomedical data analysis usually possess local, on-premise data centers with computational clusters capable of storing and processing an ample quantity of such data [46]. In contrast to the local workstations used for desktop computations, they provide an additional layer of data security, durability, reliability and even high availability. As data can be partitioned among multiple cluster nodes, many data analysis tasks can be parallelized and thus significantly accelerated. Furthermore, many such on-premise data centers are equipped with general-purpose graphics processing units (GPUs). These devices are used for the development and execution of innovative and fast algorithms to solve problems associated with genomics data analysis, such as the alignment of next generation sequencing reads [11, 19, 53], phylogenetics [6] and phylogenomics [7] and peptide or protein identification [24, 34, 37]. The primary advantage of using local data centers is convenient access to equipment that is more sophisticated than a local workstation. Data analysts typically become proficient with this equipment after some adaptation. Additionally, analysts can be certain that the computer clusters they use are always in place. This not only ensures high availability and reduces latency during data transfer but also improves data privacy and security. Moreover, this ensures that any laws requiring local data management are followed. Local data centers do have disadvantages, primarily the large capital expense that must be budgeted to acquire, maintain and upgrade the physical assets. This includes not only the equipment but also the buildings and properties. The required specifications for an on-premise data center must be estimated prior to development, meaning that an inaccurate estimation will result in either idle or limited resources.

Cloud computing responds to these disadvantages by providing computer resources (servers, storage and networking) on-demand with theoretically unlimited scaling capabilities [40]. Within this computing model, resources are available in a similar manner to residential electricity; once connected to the cloud, computational or IT resources are readily available as required. This technology is widely used in the scientific community, including in the bioinformatics domain, as demonstrated by many applications [3, 21, 23, 31, 39]. Having access to global resources is particularly relevant during the COVID-19 pandemic, with cloud computing being used to gather and process COVID-19 related data [16]. Cloud solutions have evolved over time, with traditional cloud design solutions being advanced upon by more sophisticated serverless computing models. The purpose of this paper is to review serverless computing solutions and investigate the role they can play in omics data analysis. In order to lay the foundations of this research, the next section presents an overview of cloud computing.

Cloud computing

The term ‘cloud computing’ was originally coined in 1996 by Compaq (https://www.technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/). It gained popularity 10 years later when Amazon [1] started offering Elastic Compute Cloud (EC2) services. Cloud computing is becoming increasingly specialized with regards to providing remote computing resources with the possibility of rapid provisioning [40]. This emulates, in terms of importance and contribution to the development of humanity, the introduction of the 1st power plants. Cloud service providers remove the need for users to spend time and resources on configuring and maintaining hardware and infrastructure (e.g. electricity, computer networking, cooling systems). This is a comparable step to that of electricity consumers gaining access to a national grid, and no longer being required to purchase fuel or maintain power generators.

Early cloud computing had modest functionality compared to modern implementations. Initial development saw the introduction of virtual machines (VMs), also known as virtual private servers, along with supporting features such as access key management, block storage devices and VM snapshots.

These services, together know as infrastructure as a service (IaaS), form the most basic and generic layer of the full cloud computing stack. Above this, the more abstract layers are built, such as platform as a service (PaaS), software as a service (SaaS) and more [30].

Modern cloud systems feature a wide range of services and products, including databases, business applications, business intelligence, e-commerce, quantum computing and block-chain systems. In the case of databases, both those using structured query language and those that do not (e.g. graph, document or columnar) are featured.

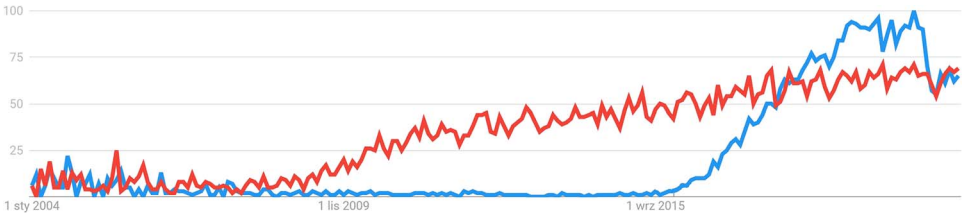

Figure 1. The increasing popularity of serverless (in blue) over IaaS (in red) computing, from Google Trends.

Cloud computing data centers have large reserves of computing power, meaning their advantages lie not only in the wide range of services provided but also in the potential scaling. Scaling in this context means response times remaining constant as the numbers of concurrent users or tasks grow. Scaling is required for mission-critical applications but is also crucial for supporting computationally complex scientific calculations [4, 38].

These scaling options can be enabled by cloud service providers across the full cloud stack. However, the ratio between implementation requirements and resultant benefits depends on the chosen layer of computing abstraction:

- Scaling IaaS requires the configuration of an external load balancer and the skills to deploy the application to be scaled on a generic operating system; nodes can be added semi-automatically, but the use of an external orchestrating tool is preferred.
- Scaling PaaS (e.g. Kubernetes, OpenShift) requires the skills to package an application into a container(s) and the use of declarative language (https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/) to define the configuration of the cluster and cluster services (i.e. applications).
- Scaling SaaS usually requires limited skills or knowledge to produce seamless results but is unable to deploy and conduct any custom computation.

Recently, the concept of serverless computing (FaaS) is becoming increasingly popular (see Figure 1). According to the Google Trends Service, global interest in this topic exceeded that of traditional IaaS in July 2018. Serverless computing hides the execution environment, allowing the user to focus on performing the relevant calculations, rather than implementation, configuration and scaling. Serverless computing features are currently available on the most popular computing clouds as services native to individual cloud service providers but also as generic platforms extending open cluster orchestration systems. Such features are highly promising as they may enable very large computing capacities for developers and scientists who do not need to be fluent with industry standards of deploying an application to the cluster.

Although most of the current applications of cloud technologies for genomics computing [5] use SaaS [8, 10, 13, 18, 26, 28, 44, 47, 52] and IaaS [3, 9, 18, 31, 32], there is growing interest in serverless implementations.

Serverless computing platforms

Traditional serverless versus containerized serverless

The FaaS concept is based on the paradigm of running a function code without dealing with problems of underlying infrastructure such as servers or VMs. Functions are designed to act in accordance with both the stateless model and the fire-and-forget model. No cloud resources are involved while no functions are activated. Functions are not designed for handling long-running processes, but they can be run as a large set of isolated instances that can be created quickly. FaaS solutions come with limitations on, for example, memory usage or execution-time of individual functions, which causes correspondingly high granularity of the division of a whole task.

The basic concept of FaaS is derived from serverless computing. A similar concept, container as a service (CaaS), supports containerization and holds several advantages resulting from serverless idea. Within CaaS, containers can be instantiated with an autoscaling option without worrying about a runtime infrastructure [5]. In the following sections, we will examine these services, with a particular focus on cloud platforms and their capabilities and limitations.

Serverless by Google

Google Cloud Platform (GCP) provides two main functionalities in terms of serverless computing: Google Cloud Functions and Google Cloud Run.

Google Cloud Functions

Google Cloud Functions (GCF) is a FaaS platform provided by Google. The functions available through GCF are primarily HTTP (synchronous mode) or event-driven (asynchronous mode). They can be triggered (https://cloud.google.com/functions/docs/concepts/events-triggers) by an HTTP protocol request, by a message available from a custom topic via the Publisher/Subscriber (asynchronous communication design pattern, https://cloud.google.com/pubsub/docs/overview) mechanism or by a message created due to changes in bucket state via a cloud storage mechanism. Several languages can be used to develop and run GCF code, including Node.js, Python, Go, C#, Java, Ruby and PHP. Direct deployment of GCF is possible via pushing source code through the GCP console or by using a zipped deployment package (e.g. using omics data analysis tools) which can be either delivered externally to GCP or be available internally via Google Cloud Storage.

The autoscaling mechanism for GCF is enabled by default. A maximum limit on the number of concurrently running GCF instances can be set. Further constraints such as the size of allocated memory (default of 256 MiB, maximum of 8 GiB) or execution timeout value (default of 1 min, maximum of 9 min) have to be set.

Google Cloud Run

Google Cloud Run (https://cloud.google.com/run) is a CaaS platform provided by Google. A container is a basic component delivering functionality that is commonly available through the HTTP representational state transfer protocol. In simple on-premise solutions (e.g. local data centers), containers are processes instantiated from a Docker image and run in traditional

one-node Docker runtime environments. In multi-node Kubernetes (K8s) environments, resources such as pods and services are used. A pod is the smallest processing unit managed by K8s which contains at least one container (most commonly only one container). The information system run on a K8s cluster of nodes is a scalable collection of running pods that are connected or exposed (also outside the K8s cluster) using services. Such a set of containers can be static, manually configurable by an administrator or modified automatically by a dedicated mechanism such as the horizontal pod autoscaler.

Google has introduced KNative (https://knative.dev/docs) to K8s—a more adaptive solution to meet requirements that include significant, unpredictable change in application load (which often happens in the analysis of biomedical data). It is dedicated to scaling the number of container instances from zero or downgrading them to zero. KNative is a CaaS solution that can be used within K8s-based systems (even those run locally). With KNative as an integrated part of K8s, cooperation between containers run by the K8s orchestrator and those spawned by KNative can be achieved. All containers may share resources available through the common K8s environment.

Since K8s is enabled as a managed service—Google Kubernetes Engine (GKE), KNative functionality is also available. As the GCP implementation of CaaS, Cloud Run can be provided through GKE (including components such as Anthos, Istio and KNative). However, Cloud Run can also be run independently of the GKE service, as a pure, fully managed CaaS, for which no handling of the K8s infrastructure is required. Both approaches provided by Cloud Run are compatible (https://cloud.google.com/anthos/run/docs/choosing-a-platform).

Required configuration parameters are a URL-based address of a container image, the size of memory to allocate, the number virtual CPUs (vCPUs) to allocate (maximum of 4) and a request timeout (default of 300 s, maximum of 1 h). The maximum value of the last parameter shows that this solution may also be useful for long-running processes, such as those performed in omics analyses. It is still however only suitable for lightweight applications, although can support heavier applications than those dedicated for FaaS. In a similar manner to GCF, Cloud Run uses autoscaling and can scale down to zero, requiring the use of containers which can be rather promptly initiated.

The following parameters control autoscaling behavior:

- a minimum number of containers (default of 0), which, for non-zero values, defines the smallest collection of running containers and prevents the number of running containers from being downgraded to zero i.e. stopping the system;
- a maximum number of containers (default of 100), which prevents uncontrolled, unrestricted and expensive increase in the size of the container set; and
- a maximum number of concurrent requests per second (default of 80), which sets the condition for detecting periods of intensive load (detecting moments of a scaling-out event).

Serverless by AWS

The Amazon Web Services (AWS) platform provides several solutions for serverless computing, including AWS Lambda and AWS Fargate. Both are used in several tools associated with omics data analysis.

AWS Lambda

In 2014, AWS introduced its FaaS platform, AWS Lambda. The functions (https://docs.aws.amazon.com/lambda/latest/dg/lambda-services.html) of AWS Lambda can be exposed by various means, including by triggers from other AWS services that can invoke lambdas synchronously (e.g. API Gateway, Application Load Balancer) or asynchronously (e.g. Message Queuing, Simple Queue Service, Simple Storage Service, Simple Notification Service, Internet of Things (IoT)).

Various languages can be used to develop AWS Lambda functions, with C#.NET, Node.js, Python, Go, Java and Ruby being supported runtime environments. Memory allocated for functions must not exceed a maximum memory size, which can be set between 128 MiB and 10 240 MiB in 1 MiB increments. Functions can be run for up to 15 min. When a function is invoked, a function instance is created. For a period of time after the function termination, the existing ready-to-use instance is available and may be reused by another invocation. Otherwise, a new instance is created. A concurrency parameter sets the maximum number of instances that can be run in parallel.

To improve deployment architecture, that is, to separate a domain component that implements a proper function and the additional required libraries, AWS introduced the concept of layers containing shared libraries. Layers may also be used to create custom runtime packages for AWS Lambda functions, which may enable the use of runtime environments or languages that are not provided by default.

In accordance with the increasing popularity of containerized solutions, the ability to deliver a function as a Docker image is also provided. It is therefore possible to use container images to deploy a function in fully managed FaaS model of AWS Lambda function processing. This is advantageous, as it allows the sharing of resources/components that can be run either in the FaaS model (AWS Lambda) or a strictly containerized model, such as Elastic Kubernetes Services (EKS), Elastic Container Services (ECS) or others based on K8s.

AWS Fargate

The AWS CaaS platform is AWS Fargate (https://docs.aws.amazon.com/AmazonECS/latest/developerguide/AWS_Fargate.html). Container-orchestration platforms can be managed by AWS. This includes those that are K8s-compatible, such as EKS, or AWS-specific solutions, such as ECS, using the AWS model of processing based on terms: container—task—service—cluster.

Using implicitly installed ECS Container Agent components, ECS may operate on a user-defined cluster of VMs (EC2 instances). A serverless alternative to ECS, AWS Fargate is a fully AWS-managed infrastructure which operates on a task-level abstraction layer (without knowledge of involved EC2 machines).

The resource limits are commonly defined at the task level. Maximum values of CPU (from 0.25 to 4 vCPUs) and memory (from 0.5 to 30 GiB) can be set. Working in the CaaS model, AWS Fargate optionally allows to use and share container image layers used by classic container solutions (EKS).

Serverless by Microsoft Azure

Azure provides many mechanisms to implement serverless computing, the most important of which are the following two building blocks.

Azure Functions

The Azure Functions service plays the most important role in the Microsoft Azure cloud when it comes to serverless computing. It is possible to use Azure Functions to build web APIs, process IoT streams, react to database changes, manage message queues, respond to file uploads and more. Durable Functions is an

extension of Azure Functions that allows the definition of stateful workflows via the development of orchestrator functions and allows the definition of stateful entities via the development of entity functions. The extension takes responsibility for managing states, checkpoints and restarts, as required.

There are several options for deploying FaaS computations which affect both the scalability and the billing method. Deployment on top of Azure Kubernetes Service (AKS) is enabled by Kubernetes-based Event Driven Autoscaling (https://docs.microsoft.com/en-us/azure/azure-functions/functions-kubernetes-keda) (KEDA) when only resources used by the Kubernetes service are required. In this case, the scaling capability is derived from a given Kubernetes cluster configuration.

Azure Functions can also be implemented via the App Service Environment (ASE). This results in a fixed monthly rate for an ASE, which pays for the infrastructure and does not change with the size of the ASE pool. There is an additional cost per vCPU used for handling function executions. The most transparent (not necessarily the cheapest) pricing plan is the Consumption Plan, under which charge is incurred based only on function run time. In this case, billing is based exclusively on the number of executions, execution time and memory used.

A further option is for billing to be dependent on vCPU core seconds used and memory consumed by required, but also pre-warmed instances. (Pre-warmed instances are instances warmed (i.e. made ready to immediately respond to a request by allocating the required computing resources in advance) as a buffer during scale and activation events. Pre-warmed instances continue to buffer until the maximum scale-out limit is reached.) At least one instance per plan must be kept warm at all times. This execution model, known as the Premium Plan, provides the most predictable pricing.

Each of the deployment options offers different network configurations, runtime environments for functions, timeout duration, scaling configuration options and cold start behavior, among other properties. These options are described in detail in the online documentation (https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale/).

Azure Container Instances

As with Google and Amazon, Microsoft offers a CaaS class solution. Azure Container Instances provides a straightforward method for the deployment of containers to Azure, freeing the user from configuration or management of lower-level services such as VPS or higher-level services such as AKS (https://azure.microsoft.com/pl-pl/services/kubernetes-service/).

Azure Container Instances allows container groups to be directly exposed to the Internet using an IP address and a fully qualified domain name. Azure Container Instances also supports command execution in a running container, providing an interactive shell for application development and troubleshooting. Access is granted over HTTPS using Transport Layer Security to secure client connections. Previously, containers offered resource management and application dependency isolation but were not sufficiently resilient for usage in multi-tenant environments. Azure Container Instances ensures, however, that the application is isolated in the container to the same extent that it would be in a dedicated VM. Containers are typically optimized to run only one application; however, the specific needs of individual applications can vary widely. Azure Container Instances ensures optimal computing resource utilization by allowing precise specifications for CPU cores and memory. Billing is set according to both demand and usage per second, so it is possible to adjust expenses efficiently, based on the needs of the user. For compute-intensive tasks such as machine learning, Azure Container Instances can schedule NVIDIA Tesla GPU resources using Linux containers.

In addition to Azure Functions and Azure Container Instances, Microsoft offers also Azure Logic Apps (https://azure.microsoft.com/en-us/services/logic-apps/), Microsoft Power Automate (https://flow.microsoft.com/en-us/) and Azure App Service WebJobs (https://docs.microsoft.com/en-us/azure/app-service/webjobs-create).

Open serverless platforms

The above solutions are services specific to individual cloud service providers, so their use will result in the vendor lock-in effect. In comparison, the KNative platform is developed on an open-source basis, but deploying it on a K8s cluster is however quite complex, and results in the creation of 110 custom resource definitions, 24 deployments, 3 daemon sets and 51 containers in total for laying foundations for functions deployment. This section details solutions that are lighter and more open.

OpenFaaS

OpenFaaS (https://www.openfaas.com/) provides the basic features expected from a serverless solution:

- avoidance of vendor lock-in thanks to the use of Docker, which is *de facto* industry standard for application containerization and deployment;
- the ability to run on any public or private cloud;
- the ability to build both microservices and functions in any language; and
- autoscaling on demand or to zero when idle.

The system architecture consists of several key components: OpenFaaS API Gateway and OpenFaaS Watchdog, which are responsible for running and monitoring functions; NATS, which, as a Cloud-native queuing system, supports asynchronous execution and queuing; and Prometheus, which provides performance metrics and enables easy autoscaling via its AlertManager.

Note that the OpenFaaS platform is suitable for use by data scientists. It provides templates for wrapping R code into serverless functions or microservices without the significant overhead that can slow deployment. A regularly updated list of templates can be found in the dedicated GitHub repository (https://github.com/analythium/openfaas-rstats-templates/).

Apache OpenWhisk

Another example of an open FaaS platform is OpenWhisk (https://openwhisk.apache.org/). This project was initially developed by IBM and is still the foundation of the IBM Cloud Functions (https://cloud.ibm.com/functions/) service. The current version is open-source, controlled by the Apache Foundation. OpenWhisk can be deployed on top of K8s, Docker Swarm, OpenShift and Mesos cluster managers. As an open-source solution, it is based on open source components, and its architecture includes an NGINX web server, which acts as an HTTP server and reverse proxy. Its main role is to support SSL (Secure Sockets Layer) encryption and the forwarding of HTTP requests to other components. The Controller component plays an important role as it uses the representational state transfer protocol API to manage the entire platform and invoke actions. All platform configuration data, function source codes and authentication and authorization data are stored in a highly

Table 1. Comparison of principal FaaS features

Service name	Supplier	Deployment/pricing models	Supported languages	License
Azure Functions	Microsoft	AKS, ASE, Consumption Plan, Premium Plan, BYO[a] (KEDA)	C#, JavaScript, F#, Java, PowerShell, Python, TypeScript	Proprietary, Apache 2.0 (KEDA)
Google Functions	Google	Native	Go, JavaScript, C#, Python, Ruby, PHP	Proprietary
AWS Lambda	Amazon	Native	C#, JavaScript, Python, Go, Java, Ruby	Proprietary
OpenFaaS	OpenFaaS Ltd.	BYO	Go, JavaScript, Python, Ruby, C#, PHP, Java	MIT
OpenWhisk	Apache	BYO	Go, JavaScript, Python, C#, PHP, Java, Rust, Scala, Swift, Ballerina	Apache 2.0

[a]'Bring Your Own', that is to say, the user must deploy on their own infrastructure or IaaS.

scalable CouchDB document database. As a request queuing system, it uses the Kafka message broker. All requests from the Controller are handled by Kafka and then forwarded to corresponding Invokers. Relying on Docker, these Invokers form the core of OpenWhisk.

The FaaS solutions discussed in this chapter are listed in Table 1, which specifies the deployment models, supported programming languages and licenses. Many other serverless computing orchestration platforms are currently under active development, such as Kubeless (https://github.com/kubeless/kubeless/), Fission (https://github.com/fission/fission/) and Fn (https://github.com/fnproject/fn/). Serverless computing is an area of active development, with new solutions expected to add to those listed here.

Omics serverless computing

Given the diversity of platforms and approaches to delivering serverless computing, developers have a high degree of choice in picking the best solution to a particular scenario of omics data analysis. However, the choice may be limited by the availability and the runtime environment of particular tools that must be integrated while implementing a complex omics data analysis pipeline. In this section, we review several case studies of serverless computing being used in various areas of bioinformatics that involve the approaches described in the previous section.

Case studies

Challenges and potential opportunities of using AWS Lambda functions for running bioinformatics workflows were described by Crespo-Cepeda *et al.* [15]. The authors consider an application that mines single nucleotide polymorphism (SNP) data. Rules are extracted from the data which allow patients to be associated with their responsiveness to certain drugs. The authors proposed the use of a serverless architecture that takes advantage of AWS Lambda and AWS S3 as a storage engine. The application itself is based on DMET-Miner, adapted for AWS Lambda runtime. Within the proposed architecture, processing is initiated upon uploading a specific task file to an S3 bucket, which in turn triggers a serverless function to interpret the task and launch asynchronous worker functions that download the SNP dataset and execute the DMET-Miner workflow. The outputs of each worker function are saved to a separate directory on the S3 Bucket. The SNP dataset is also stored on the same bucket. The authors ran performance experiments where they observed that, for certain scenarios, processing time could be reduced from 2 h and 20 min of synchronous processing to 2 min and 8 s of parallel processing with AWS Lambda.

Aboukhalil [2] presents another example of the use of serverless computing in bioinformatics. The author built an API for simulating DNA sequencing data, in the form of a serverless function running on Cloudflare Workers Unbound [14]. The solution is based on 'wgsim', an existing tool for simulating sequence reads from a reference genome. Using the 'biowasm' recipes, 'wgsim' was compiled from C to WebAssembly. As a reference DNA sequence, the author used a publicly accessible human genome reference dataset taken from the 1000 Genomes Project, which is part of the AWS Open Data Registry [43] and is hosted on an AWS S3 bucket. As the dataset was over 3 GBs in size, and due to the limited memory available for serverless functions, the solution only fetched subsets of reference data specified as parameters and streamed the simulation results back to the user. The solution also offers a simple graphical interface for more interactive exploration of simulation results.

Another tool that relies on serverless architecture has been proposed by Lee *et al.* [33]. The authors present the DNAVisualisation (https://dnavisualisation.org/) application for DNA sequence visualization. The application takes advantage of the function-as-a-service model to parallelize the visualization of DNA sequences. Data are accepted in the FASTA format, then parsed and submitted in parallel to serverless functions, which transform the sequences and then store them in Parquet format on an AWS S3 bucket. While the user explores the data (e.g. zooms or rearranges the visualization), processing is handled by a serverless function that retrieves all necessary data from previously transformed Parquet files, using S3 Select. According to the authors, the application is limited to visualizing up to 30 sequences, each with a maximum size of 4.5 MB. There are five supported visualization methods: 'Yau', 'Squiggle', 'Randić', 'Qi' and 'Gates'. The application also offers the ability to export data in PNG, SVG, JPEG or JSON format. The solution is based on the AWS Lambda offering, but the authors note that it can be ported to other cloud providers such as Azure or Google Cloud.

Portability across different cloud platforms was tested by Niu *et al.* [41], who proposed and described an implementation of all-against-all pairwise comparison between over 20 000 human protein sequences. The authors used the striped Smith–Waterman algorithm [17] to perform the comparison, running on AWS Lambda and GCF platforms. To take advantage of the massive parallelism capabilities of these platforms, the authors partitioned protein sequences into subsets of 500 sequences.

They identified that individual functions, given the constraints of the environment, would be able to compute the interactions between two such subsets successfully. This resulted in 861 unique tasks being executed. After running benchmarking experiments, the authors observed that running the workload on Lambda functions triggered directly from a laptop took on average 2.2 min; for GCF, it was about 11.82 min, while running the whole workload on a laptop computer took almost 9 h. For both AWS and Google-based solutions, the cost of running sequence comparison was less than $1.

The parallelization of data processing and analysis is an inherent property of serverless computing that provides a reduction in computation time. This was demonstrated by Hung *et al.* [25], who presented a serverless approach to the analysis of RNA sequencing data, in order to determine differentially expressed genes. The authors proposed a three-step architecture: split, merge and align. The merge step is parallelized with serverless functions that align human transcriptome reads by using the Burrows–Wheeler Aligner [35]. The other two steps are executed from a laptop computer, which also triggers functions that execute alignment workflow. During implementation, the authors had to accommodate constrained computing capabilities and limited execution time of a single function invocation. They managed to overcome these challenges by reducing processed file sizes and using asynchronous execution with partial data, along with configuration tuning and optimization. With their configuration, they determined that sharding data into files with a size of approximately 60 MB was near-optimal, which translated to a workflow invoking 1752 functions in parallel during the alignment step. By taking advantage of such an architecture, they were able to reduce the total execution time of the workflow from 2.5 h when running on a cloud server with 16 threads to about 6 min, with subsequent workflows taking less than 2 min to obtain results. The estimated cost of running the whole workflow was less than $4.

A slightly different approach was adopted by Burkat *et al.* [12], who compared the feasibility of using CaaS platforms such as AWS Fargate and Google Cloud Run, to FaaS offerings such as AWS Lambda, as potential computing platforms for running scientific workflows. The authors extended the HyperFlow (https://github.com/hyperflow-wms/hyperflow) engine by adding support for the above platforms. Three workflows were considered: AutoDock Vina [51], which is an open-source tool used for molecular docking and virtual screening; OSG-KINC [45], an open-source application that is used to construct gene co-expression networks based on Open Science Grid resources; and Soybean Knowledge Base (SoyKB) [29], which is an application used for soybean translational genomics. Due to the complexity of the SoyKB workflow, the authors proposed a hybrid approach that uses both AWS Lambda and AWS Fargate. The authors highlight the fact that serverless computing can be used for complex workflows and that a combination of AWS Lambda and AWS Fargate for specific parts of the workflow can be more effective than using either alone. They also list AWS Fargate as more suitable for long-running tasks than AWS Lambda. The authors note that AWS Fargate could be replaced with a cluster of VMs, which would however introduce additional maintenance and setup overhead.

Reducing execution time and usage costs for cloud infrastructure provides a large motivation for serverless computing. The Commonwealth Scientific and Industrial Research Organisation (https://www.csiro.au/) (CSIRO), one of the strongest proponents for the adoption of serverless architectures for bioinformatic workflows, notes however that many workloads implemented for omics data analysis have an unpredictable nature. Many serverless architectures have been developed by CSIRO. One example is their GT-Scan suite (https://gt-scan.csiro.au/), a web application that finds targets with minimal similar sequences in their genome. Since the application must always be available but may not always be in use, CSIRO proposed a solution based on AWS Lambda and AWS DynamoDB, which allowed costs per month to be reduced from $714 for a VM-based solution to around $2.50 [48]. Another use case developed at CSIRO is sBeacon [27], which is a serverless implementation of the Beacon protocol (https://beacon-project.io/), an open protocol for the discovery of genomics data, used, for example, for detecting mutations that cause diseases. Reimplementation on AWS Lambda and AWS S3 reduced the time taken to upload new genomes into the database—in one case, it reduced the time from 33 h for an AWS EC2-based solution to only 22 s. In addition, both query run time and costs were reduced, as information has to be collected only from relevant parts of the database, which was previously stored in separate files across AWS S3 buckets. The authors also state that the chosen architecture helped to ensure privacy and ownership of data. They also prepared a supplementary repository dedicated to the use of sBeacon in the context of rapid querying of the SARS-CoV-2 genome (https://github.com/aehrc/COVID-sBeacon). In their latest research, the authors used the Serverless Variant Effect Predictor (https://github.com/aehrc/sVEP) (sVEP) to predict genomic variants using a serverless architecture. The sVEP analyzes and predicts genomic variants, which in turn can be used to select improved treatment for patients. Due to potential parallelization of the workflow thanks to the AWS Lambda scaling model, sVEP has been estimated to be 99% faster than traditional VEP implementations [49]. It was developed at CSIRO in collaboration with Pathology Queensland and QIMR Berghofer (https://www.qimrberghofer.edu.au/).

A further example of a serverless approach in bioinformatics was presented by Grzesik *et al.* [20]. The authors propose, implement and evaluate the feasibility of running a basecalling process for nanopore sequencing reads. They use AWS Lambda as the underlying computing platform due to its computing capabilities and support for Docker containers. During experiments, the authors evaluated several basecalling tools. They determined that three to four Lambda functions have enough computing power to support near real-time processing of data from a single MinION Nanopore (https://nanoporetech.com/products/minion) device and that it is possible to scale up to 100 simultaneously running functions in less than 1 min. The authors also suggest that thanks to recent improvements to AWS Lambda, it is a more promising choice for the growing number of bioinformatics applications.

The findings of this section are summarized in Table 2.

Commonly used computing architecture

Several of the case studies in Section 4.1 share a similar architectural approach to taking advantage of serverless computing, specifically regarding the massive parallelism that it offers. The authors of the works identify the section of the considered workflow that is most computationally demanding and replace it with a three-step process. In the 1st step, data are prepared and chunked into smaller batches. This data preparation process is driven by the limitations on computing power available to a single serverless function, as well as the limited execution time, for example, only 15 min for AWS Lambda runtime. Following the 1st step, the 2nd step is triggered, in which multiple functions, running in parallel, process previously prepared data and output

Table 2. Summary of omics serverless computing case studies

System	Platform(s)	Workflow	Algorithms/tools	Benefits of using serverless computing
Crespo-Cepeda [15]	AWS Lambda	SNP genotyping data mining	DMET-Miner	Reduced processing time
Aboukhalil [2]	Cloudflare Workers	Sequence read simulation	wgsim	Cost-effectiveness
Lee *et al.* [33]	AWS Lambda	DNA sequence visualisation	Squiggle	Parallelization, scalability, cost-effectiveness
Niu *et al.* [41]	AWS Lambda, GCF	All-against-all protein comparison	Striped Smith-Waterman	Reduced processing time, cost-effectiveness
Hung *et al.* [25]	AWS Lambda, GCF	Differentially expressed gene determination	BWA	Reduced processing time, cost-effectiveness
Burkat *et al.* [12]	AWS Fargate, AWS Lambda	Molecular docking	AutoDock Vina	Reduced maintenance
Burkat *et al.* [12]	AWS Fargate, Google Cloud Run	Gene co-expression network construction	OSG-KINC	Reduced maintenance
Burkat *et al.* [12]	AWS Fargate, AWS Lambda	Genotypic data analysis	SoyKB	Reduced maintenance
Grzesik *et al.* [20]	AWS Lambda	Nanopore read basecalling	Guppy, Bonito	Parallelization of processing
sBeacon [27]	AWS Lambda	Genomics data discovery	Beacon protocol	Reduced processing time, improved privacy
GT-Scan2 [48]	AWS Lambda	Genomic sequence target finding	n/a	Cost-effectiveness
sVEP [49]	AWS Lambda	Genomic variants prediction	n/a	Reduced processing time

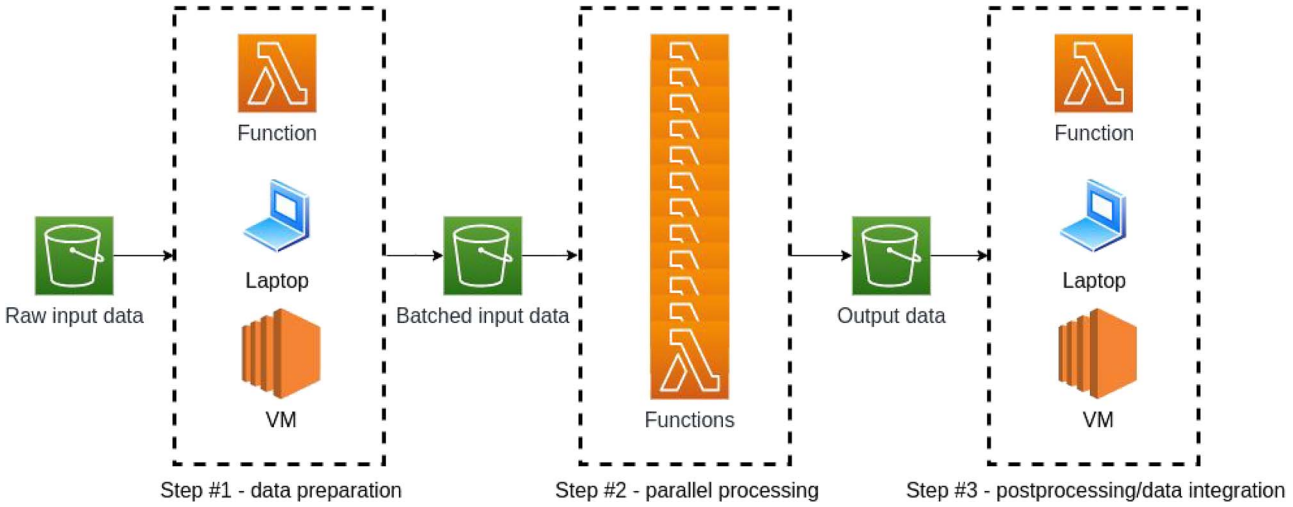

Figure 2. Diagram of commonly used architecture for serverless bioinformatics processing.

the results to shared storage. In the optional 3rd step, after all functions from the 2nd step finish processing, data can be integrated to produce final results. In all cases, the data for all steps are stored in shared, cloud-based object storage such as AWS Simple Storage Service, Google Cloud Storage or Microsoft Azure Blob Storage. Although the 1st and 3rd steps can be executed by a serverless function, local laptop or a VM running in the cloud, the 2nd step always runs on top of a serverless platform. The general architecture described above is presented in Figure 2.

Discussion

Since its emergence in the past decade, serverless computing has become a mature technology capable of delivering the computing power required to perform scientific calculations associated with data processing and analysis. Serverless computing has developed important properties that determine its application possibilities. It is capable of delivering computational solutions that can be scaled automatically as required. This is important functionality for multi-omics data analysis due to the increasing and unpredictable growth of data. Furthermore, serverless computing is also event-driven, which allows for the execution of data analysis in response to incoming events. For example, a machine learning model for data clustering may perform successive steps of data analysis. Serverless computing also eliminates operational overhead associated with the maintenance of node farms in a data analysis cluster. This simplifies the preparation of the computational infrastructure and allows the preparation of the analysis pipeline to be the primary focus. Consequently, serverless computing allows for faster adaptation to the current demands of data analysis. This property has been especially valuable in the case of the COVID-19

pandemic, which occurred suddenly and did not allow scientists to prepare their current infrastructures for new data sets and new challenges, but required integration of new data sources and new types of data into analysis pipelines. Moreover, serverless computing provides many built-in services for tool and data source integration, which is especially useful in situations which require fast adaptation, such as the pandemic. Unlike elements of the native cloud infrastructure, under the serverless computing model, it is not possible to over-provision the computational resources required for particular data analysis workflows. It is also worth noting that serverless computing platforms are constantly expanding and offering more capabilities. For example, AWS Lambda recently added support for Docker container images, which in the future might unlock new bioinformatics use cases and workflows. These properties have resulted in serverless computing being used by several research teams focused on multi-omics data analysis, suggesting that this new medium for performing computations may also hold advantages for the bioinformatics community.

Although serverless computing can offer substantial benefits, it also comes with several drawbacks and challenges that must be considered when implementing a new solution or migrating an existing one. The major constraint is limited computing capability regarding available vCPUs and memory per single function or container. Another challenge is introduced by limited execution time. For example, in the case of GCF, the maximum function timeout is only 9 min. The implementation process is often driven by both of these factors, as each step in an implemented workflow has to execute in a constrained environment in a limited period of time, which often requires data to be split into multiple chunks, so that each chunk can be successfully processed given the above constraints. This approach is presented in Figure 2. However, this technique is not always applicable. If a certain process requires, for example, all data to be loaded into memory for processing, the limited memory available to a function will not be sufficient. Such tasks must be executed elsewhere, for example, on a traditional VM cluster. When it comes to currently available computing capabilities, support for GPU acceleration is very limited, with only Azure Container Instances offering deployments with access to GPU resources. Nonetheless, given the rapid growth of serverless platforms, it is highly likely that more providers will begin to offer similar support. Although local development can be more challenging than for traditional approaches, it has become increasingly straightforward due to support for container images, which can also be run and tested locally. When choosing a serverless platform, it should be noted that offerings from major cloud providers such as AWS, Microsoft Azure or GCP can introduce vendor lock-in, making future migration to other platforms more difficult. That risk can however be mitigated by taking advantage of containers and/or open-source platforms such as OpenFaaS or Apache OpenWhisk.

Conclusions

Serverless computing is increasingly popular and is slowly becoming an architectural element of integrative data analysis in bioinformatics. The technology has the potential to be used in COVID-19 related projects which demand access to large and varied types of globally gathered data.

Ongoing development of the technology along with the emergence of new services may accelerate adoption of the new computing model in created analysis environments. The use cases presented in Section 4.1 demonstrate that serverless computing can be successfully used for performing multi-omics data analyses and that, compared to alternative computing approaches, it can provide benefits such as reduced processing time, cost-effectiveness, reduced maintenance overhead, parallelization of processing and improved privacy of processed data.

The use cases further demonstrate the applicability and usefulness of the model in the newly designed computer architectures for bioinformatics. It is the belief of the authors of this work that future improvements in serverless computing will enable additional bioinformatic workflows that can take advantage of such architectures.

Key Points

- Serverless computing simplifies operations associated with multi-omics data analysis in cloud environments.
- Various serverless approaches determine the way of reusing the existing bioinformatics tools and packages.
- Hiding the complexity of deployment and software scaling from end-users makes the parallelization of complex analysis pipelines much easier.
- By using function-as-a-service solutions, cloud computing costs may be reduced for infrequent multi-omics calculations.
- Development of time-critical multi-omics data analysis pipelines for COVID-19 patients could be accelerated by using serverless computing.

Acknowledgments

This work was supported by the professorship grant (02/020/RGP19/0184) of the Rector of the Silesian University of Technology, Gliwice, Poland, the pro-quality grant for highly scored publications or issued patents of the Rector of the Silesian University of Technology, Gliwice, Poland (grant No 02/100/RGJ21/0009), the Statutory Research funds of Department of Applied Informatics, Silesian University of Technology, Gliwice, Poland (grant No 02/100/BK_21/0008), and the Polish Ministry of Science and Higher Education as a part of the CyPhiS program at the Silesian University of Technology, Gliwice, Poland (Contract No. POWR.03.02.00-00-I007/17-00).

References

1. Amazon AWS Documentation. *Announcing Amazon Elastic Compute cloud (Amazon EC2)—Beta*. https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2—beta/ (23 August 2021, date last accessed).
2. Aboukhalil R. *Serverless Genomics—Using WebAssembly and Cloudflare Workers to Power Genomics Analysis*. https://robaboukhalil.medium.com/serverless-genomics-c412f4bed726 (5 June 2021, date last accessed).
3. Angiuoli SV, Matalka M, Gussman A, *et al*. CloVR: a virtual machine for automated and portable sequence analysis from the desktop using cloud computing. *BMC Bioinformatics* 2011;**12**:356.

4. Anwar N, Deng H. Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. *Future Internet* 2018;**10**(1):5.
5. Augustyn DR, Wyciślik L, Mrozek D. Perspectives of using Cloud computing in integrative analysis of multi-omics data. *Brief Funct Genom* 2021;**20**(4):198–6.
6. Ayres DL, Darling A, Zwickl DJ, *et al*. BEAGLE: an application programming interface and high-performance computing library for statistical phylogenetics. *System Biol* 2011;**61**(1):170–3.
7. Baele G, Ayres DL, Rambaut A, *et al*. *High-Performance Computing in Bayesian Phylogenetics and Phylodynamics Using BEAGLE*. New York, NY: Springer, 2019, 691–722.
8. Bi JH, Tong YF, Qiu ZW, *et al*. ClickGene: an open cloud-based platform for big pan-cancer data genome-wide association study, visualization and exploration. *BioData Min* 2019;**12**(12):1–15.
9. Birger C, Hanna M, Salinas E, *et al*. FireCloud, a scalable cloud-based platform for collaborative genome analysis: strategies for reducing and controlling costs. *bioRxiv* 2017; **209494**:1–28.
10. Blatti C, 3rd, Emad A, Berry MJ, *et al*. Knowledge-guided analysis of "omics" data using the KnowEnG cloud platform. *PLoS Biol* 2020;**18**(1):e3000583–3.
11. Buntara F, Lee B, Purbojati RW, Zhou CX. Is GPUs ready to boost genomic alignment computation. *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*, Aswan, Egypt: IEEE Press, 2019, 130–5. doi: 10.1109/ITCE.2019.8646637.
12. Burkat K, Pawlik M, Balis B, *et al*. Serverless containers—rising viable approach to scientific workflows. ArXiv, abs/2010.11320, 2020, pp. 1–13.
13. Chervova O, Conde L, Afonso Guerra-Assunção J, *et al*. The personal genome project-UK: an open access resource of human multi-omics data. *Sci Data* 6, **257** (2019), pp. 1–10. https://doi.org/10.1038/s41597-019-0205-4.
14. Gao N. *Announcing Cloudflare Workers Unbound for General Availability*. https://blog.cloudflare.com/workers-unbound-ga/ (5 June 2021, date last accessed).
15. Crespo-Cepeda R, Agapito G, Vazquez-Poletti JL, *et al*. Challenges and opportunities of Amazon serverless Lambda services in bioinformatics. In: *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, BCB '19*. New York, NY, USA: Association for Computing Machinery, 2019, 663–8.
16. Farah I, Lalli G, Baker D, *et al*. A global omics data sharing and analytics marketplace: case study of a rapid data COVID-19 pandemic response platform. medRxiv, 2020.
17. Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics* 2006;**23**(2):156–61.
18. Feng X, Grossman R, Peakranger LS. A cloud-enabled peak caller for chip-seq data. *BMC Bioinformatics* 2011;**12**(1): 139.
19. Frohmberg W, Kierzynka M, Blazewicz J, *et al*. G-DNA—a highly efficient multi-GPU/MPI tool for aligning nucleotide reads. *Bull Pol Acad Sci* 2013;**61**(4):989–92.
20. Grzesik P, Mrozek D. Serverless nanopore basecalling with AWS Lambda. In: Paszynski M, Kranzlmüller D, Krzhizhanovskaya VV, *et al*. (eds). *Computational Science—ICCS 2021*, Vol. 12743. Cham: Springer International Publishing, 2021, 578–86.
21. Heath AP, Greenway M, Powell R, *et al*. Bionimbus: a cloud for managing, analyzing and sharing large genomics datasets. *J Amer Med Inform Assoc* 2014;**21**(6):969–75.
22. Hull D, Wolstencroft K, Stevens R, *et al*. Taverna: a tool for building and running workflows of services. *Nucleic Acids Res* 2006;**34**(suppl_2):W729–32.
23. Hung C-L, Chen W-P, Hua G-J, *et al*. Cloud computing-based tagsnp selection algorithm for human genome data. *Int J Mol Sci* 2015;**16**(1):1096–110.
24. Hung C-L, Lin Y-S, Lin C-Y, *et al*. CUDA ClustalW: an efficient parallel algorithm for progressive multiple sequence alignment on multi-GPUs. *Comput Biol Chem* 2015;**58**:62–8.
25. Hung L-H, Niu X, Lloyd W, *et al*. Accessible and interactive rna sequencing analysis using serverless computing. bioRxiv, 2020.
26. Ivanov AA, Revennaugh B, Rusnak L, *et al*. The OncoPPi Portal: an integrative resource to explore and prioritize protein-protein interactions for cancer target discovery. *Bioinformatics* 2017;**34**(7):1183–91.
27. Jain Y, Hosking B, Twine N, *et al*. sBeacon: cloud-native genomic data exchange. *ABACBS-2020* 2020;**2020**(1):11.
28. Jalili V, Afgan E, Gu Q, *et al*. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2020 update. *Nucleic Acids Res* 2020;**48**(W1):W395–402.
29. Joshi T, Patil K, Fitzpatrick M, *et al*. Soybean Knowledge Base (SoyKB): a web resource for soybean translational genomics. *BMC Genom* 2012;**13**(Suppl 1):S15.
30. Kavis M. *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)*. Wiley CIO Series. Hoboken, New Jersey: Wiley, 2014.
31. Krampis K, Booth T, Chapman B, *et al*. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC Bioinformatics* 2012; **13**:42.
32. Langmead B, Nellore A. Cloud computing for genomic data analysis and collaboration. *Nat Rev Genet* 2018;**19**(4):208–19.
33. Lee BD, Timony MA, Ruiz P. DNAvisualization.org: a serverless web tool for DNA sequence visualization. *Nucleic Acids Res* 2019;**47**(W1):W20–5.
34. Li C, Li K, Li K, *et al*. MCtandem: an efficient tool for large-scale peptide identification on many integrated core (MIC) architecture. *BMC Bioinformatics* 2019;**20**(397):1–13.
35. Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 2009;**25**(14): 1754–60.
36. Li Y, Hou G, Zhou H, *et al*. Multi-platform omics analysis reveals molecular signature for COVID-19 pathogenesis, prognosis and drug target discovery. *Sig Transduct Target Ther* 2021;**6**(155):1–11.
37. Li Y, Chu X. Speeding up scoring module of mass spectrometry based protein identification by GPU. In: *2012 IEEE 14th International Conference on High Performance Computing and Communication, 2012 IEEE 9th International Conference on Embedded Software and Systems*, Liverpool, UK: IEEE Press, 2012, 1315–20.
38. Lin B, Zhu F, Zhang J, *et al*. A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. *IEEE Trans Ind Inf* 2019;**15**(7): 4254–65.
39. Masseroli M, Canakoglu A, Pinoli P, *et al*. Processing of big heterogeneous genomic datasets for tertiary analysis of next generation sequencing data. *Bioinformatics* 2018;**35**(5): 729–36.

40. Mell PM, Grance T. Sp 800-145. The NIST definition of cloud computing. Technical Report. National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, USA, 2011.
41. Niu X, Kumanov D, Hung L-H, *et al*. Leveraging serverless computing to improve performance for sequence comparison. In: *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, BCB '19*. New York, NY, USA: Association for Computing Machinery, 2019, 683–7.
42. Oinn T, Addis M, Ferris J, *et al*. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 2004;**20**(17):3045–54.
43. *Registry of Open Data on AWS*. https://registry.opendata.aws/ (5 June 2021, date last accessed).
44. Patel RY, Shah N, Jackson AR, *et al*. Clingen pathogenicity calculator: a configurable system for assessing pathogenicity of genetic variants. *Genome Med* 2016;**9**, pp. 1–9.
45. Poehlman WL, Rynge M, Balamurugan D, *et al*. OSG-KINC: High-throughput gene co-expression network construction using the open science grid. In: *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Kansas City, MO, USA: IEEE Press, 2017, 1827–31.
46. Psiuk-Maksymowicz K, Jaksik R, Placzek A, *et al*. Biotest—remote platform for hypothesis testing and analysis of biomedical data. In: Korbicz J, Maniewski R, Patan K, *et al*. (eds). *Current Trends in Biomedical Engineering and Bioimages Analysis*. Cham: Springer International Publishing, 2020, 152–65.
47. Qu K, Garamszegi S, Wu F, *et al*. Integrative genomic analysis by interoperation of bioinformatics tools in GenomeSpace. *Nat Methods* 2016;**13**(3): 245–7.
48. Bauer D, Hosking B. *What Is "Serverless" and "Cloud-Native" and When to Use It?* https://bioinformatics.csiro.au/blog/converting-traditional-architecture-to-cloud-native-applications/ (5 June 2021, date last accessed).
49. Transformational Bioinformatics. *Serverless VEP*. https://bioinformatics.csiro.au/serverless-vep/ (5 June 2021, date last accessed).
50. Tomazou M, Bourdakou MM, Minadakis G, *et al*. Multi-omics data integration and network-based analysis drives a multiplex drug repurposing approach to a shortlist of candidate drugs against COVID-19. *Brief Bioinform* 2021: bbab114.
51. Trott O, Olson AJ. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *J Comput Chem* 2010;**31**(2):455–61.
52. Wang L, Yang L, Peng Z, *et al*. cisPath: an R/Bioconductor package for cloud users for visualization and management of functional protein interaction networks. *BMC Syst Biol* 2015;**9**(1):S1.
53. Wilton R, Szalay AS. Arioc: high-concurrency short-read alignment on multiple GPUs. *PLoS Comput Biol* 2020;**16**(11): 1–11.