In-silico Research Platform in the Cloud - Performance and Scalability Analysis

1st Miloš Ivanović
Faculty of Science
University of Kragujevac
BIOIRC d.o.o.
Kragujevac, Serbia
0000-0002-8974-2267

2nd Andreja Živić
Faculty of Science
University of Kragujevac
BIOIRC d.o.o.
Kragujevac, Serbia
andrejazivic2@gmail.com

3rd Nikolaos Tachos
Dept. of Materials Science and Engineering
University of Ioannina
Ioannina, GR45110, Greece
0000-0002-8627-6352

4th George Gois
Dept. of Materials Science and Eng.
University of Ioannina
Ioannina, GR45110, Greece
gkois@yahoo.com

5th Nenad Filipović
Faculty of Engineering
University of Kragujevac
BIOIRC d.o.o.
Kragujevac, Serbia
0000-0001-9964-5615

6th Dimitrios I. Fotiadis
Dept. of Materials Science and Eng.
University of Ioannina
Ioannina, GR45110, Greece
fotiadis@cc.uoi.gr

***Abstract*—The paper describes experiences from building and cloudification of the in-silico research platform SilicoFCM, an innovative in-silico clinical trials' solution for the design and functional optimization of whole heart performance and monitoring effectiveness of pharmacological treatment, with the aim to reduce the animal studies and the human clinical trials. The primary aim of cloudification was to prove portability, improve scalability and reduce long-term infrastructure costs. The most computationally expensive part of the platform, the scientific workflow manager, was successfully ported to Amazon Web Services. We benchmarked the performance on three distinct research workflows, each of them having different resource requirements and execution time. The first benchmark was pure performance of running workflow sequentially. The aim of the second test was to stress-test the underlying infrastructure by submitting multiple workflows simultaneously. The benchmark results are promising, painting the infrastructure launching overhead almost negligible in this kind of heavy computational use-case.**

***Index Terms*—scientific workflow, cloud computing, in-silico platform, scalability, portability**

I. Introduction

Biomedical and bio-engineering research is becoming increasingly dependent on secure and scalable services for computing, storage, and networking. Traditionally, these infrastructural elements are deployed and maintained on-premises of research institutions. Recently, for various kinds of biomedical applications, cloud computing has emerged as an alternative to locally maintained infrastructures. Cloud computing services offer secure on-demand computing, storage, and platform services which are clearly differentiated from high-performance computing by their rapid availability and scalability. As such, cloud services are successfully addressing a large class of biomedical problems and enhance the likelihood of data and workflows sharing, reproducibility, and reuse.

There are a lot of examples of successful cloud platforms for biomedical research. For example, Galaxy, an open-source, web-based scientific workflow platform, is used for data–intensive biomedical research [1]. For large-scale data analysis, Galaxy can be hosted in cloud IaaS, as described in [18]. Reliable and highly scalable cloud-based workflow systems for next-generation sequencing analyses have been achieved by integrating the Galaxy workflow system with Globus Provision [4]. The Bionimbus Protected Data Cloud (BPDC) is a private cloud-based infrastructure for managing, analyzing, and sharing large amounts of genomics and phenotypic data in a secure environment, which was used for gene fusion studies [2]. BPDC is primarily based on OpenStack[1], open-source software that provides tools to build cloud platforms, with a service portal for a single point of entry and a single sign-on for various available BPDC resources.

The complete cloud platforms nowadays provide an environment for establishing an end-to-end pipeline for data acquisition, storage, and analysis. For example, Seven Bridges Genomics (SBG) offers both genomics SaaS and PaaS and employs AWS as backend. SBG Platform also enables researchers to collaborate on the analysis of large cancer genomics data sets in a reproducible and scalable manner. They base all their workflow management on a standardized Common Workflow language (CWL) to facilitate developers, analysts, and biologists to deploy, customize, and run reproducible analysis methods. Users may choose from over 200 tools and workflows covering many aspects of genomics data processing to apply to TCGA data or their own data sets. A lot more

SilicoFCM H2020 funding

[1]https://www.openstack.org/

examples of successful biomedical cloud platforms can be found in [5].

Public clouds also provide batch processing capabilities (AWS Batch, EKS and ECS, etc.) that automatically provision the optimal quantity and type of compute resources based on the volume and specific resource requirements of the batch jobs submitted, thereby significantly facilitating analysis at scale.

For practical reasons, but also to promote open-science approach, all data, analytical tools and methods should be findable, accessible, inter-operable and reusable (FAIR principles). The FAIR principles serve as a guideline for data producers and researchers to be interoperable as much as possible. The individual tools are organized and interconnected in a standardized way. This automatically implies packaging software using Linux container technologies, such as Docker or Singularity[2], and then orchestrating workflows and pipelines using domain-specific workflow language such WDL (Workflow Description Language) and CWL (Common Workflow Language).

In this paper, we provide the experience of porting SilicoFCM research platform[3] to AWS and discuss different aspects of such undertake. The paper is organized as follows. After the introduction, the second section describes hardware and virtualization layers of the platform hosted locally. The third section provides specifics of AWS deployment and workflow management in that environment. The benchmark results are given and discussed in Section IV.

II. Local deployment of the platform

At the beginning of the platform development, the decision has been made to make SilicoFCM platform portable across different virtualization providers, including public clouds. The development has begun using the infrastructure on premises of one of the partners.

A. Hardware base

Physical servers are deployed in a physically isolated rack, which belongs to the isolated network domain. It consists of 3 hypervisor nodes (32 cores, 128GB RAM) aggregated into a Proxmox VE cluster, one storage server, and a dedicated hardware router/firewall providing VPN access. The above sums up to 108 CPU cores, 416 GB RAM, and 22 TB raw storage, interconnected with 10 Gbps internal network, 1 Gbps management network, and 1 Gbps uplink.

As depicted in Figure 1, a cluster management network (red lines) acts as a separate VLAN (Virtual LAN) to the data network (blue lines). The storage is accessible only from the internal network. This architecture decision provides an additional layer of data security.

[2]https://www.docker.com/, https://sylabs.io/singularity/

[3]https://silicofcm.eu/

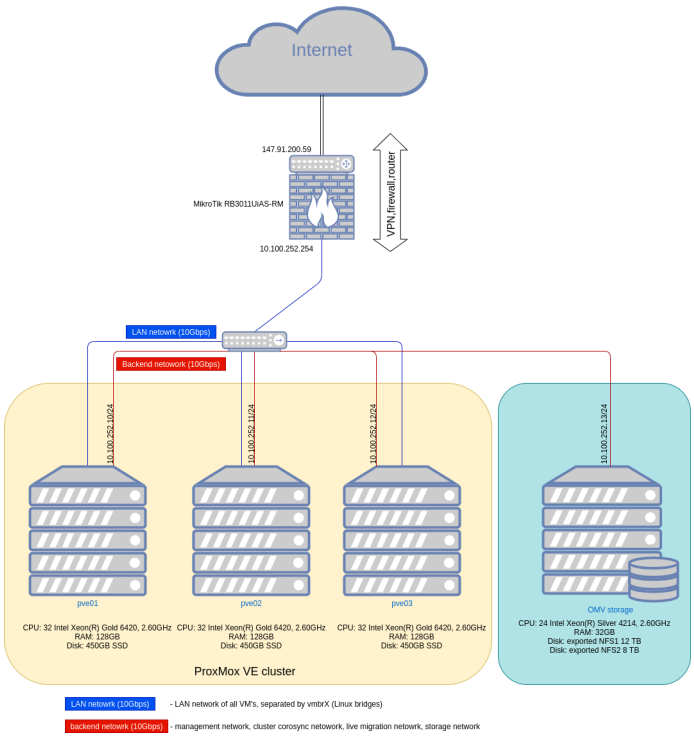

Fig. 1. Hardware foundations of the SilicoFCM platform

B. Logical architecture

Two separate platform deployments have been established from the very beginning: *Development* and *Testing*. *Development* platform is the fast-changing one and acts as an upstream for *Testing*. Figure 2 depicts the logical architecture of the platform. It consists of:

- Reverse HTTP(S) proxy,
- Nodes belonging to the *Development* group,
- Nodes belonging to the *Testing* group,
- *Common services* used by both *Development* and *Testing*.

NodeWebApp is a web application server based on NodeJS framework. *DBServer* provides SQL and NoSQL services to the other modules, while *AuthServer* provides `OAuth2` compatible authentication. The most resource-demanding component is *FunctionalEngineServer*, responsible for the execution of the CWL[4] compatible scientific workflows. As stated above, the virtual machines that make up the *Development* deployment are identical to their counterparts in *Testing*.

The services that belong to the *Common Services* group are already well-established, adopted, and thoroughly tested modules. They do not require a separate development branch due to a slower pace of development compared to the core SilicoFCM modules.

C. Platform security

Besides the hardware firewall, the SilicoFCM platform makes use of the reverse proxy to fine-tune the OSI Layer 7

[4]Common Workflow Language - https://www.commonwl.org/

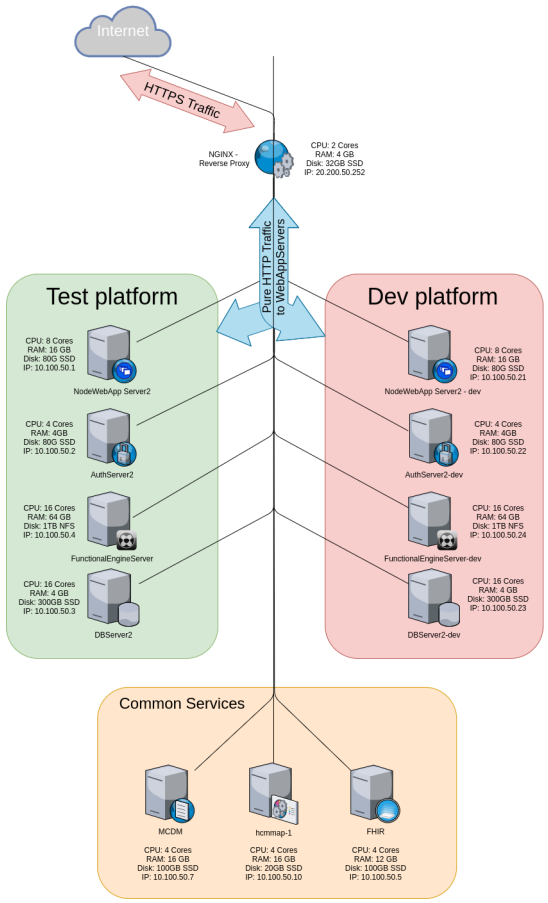

Fig. 2. Logical architecture of the SilicoFCM platform

access to the platform services. Nginx[5] in reverse proxy mode is employed to control all HTTP(S) traffic from the Internet, as depicted in Fig. 2. Due to the strict security requirements during the development phase, but still, to provide a sufficient degree of user-friendliness, Nginx was configured to provide limited access based on the GeoIP approach. This policy is subject to change in the future releases. The gateway also provides SSL support for the whole platform.

III. The Elements of the AWS Deployment

As stated above, the complete platform twin has also been ported on Amazon Web Services. The major motivations for this porting effort were to:

- **Test and improve portability** of the platform components and the platform as a whole.
- **Provide sufficient computational scalability** for the platform users. Running multiple workflows by multiple users simultaneously can cause bottlenecks.
- **Provide sufficient storage capacity**. Despite 22 TB raw storage available locally, heavy platform usage can lead to a lack of storage space, especially on FES permanent and temporary storage, but also the back-end file manager based on Kitware Girder[6].
- **Automate resource provisioning**, including both virtual machines and storage space to avoid unnecessary IaaS costs. Since the platform's demands are quite high, leaving "dangling" resources can cause significant issues.

Starting from the scheme shown in Figure 2, the specific architectural solution for AWS resulted in the scheme shown in Figure 3. It employs the following services offered by AWS:

- **AWS EC2** - Creating, managing, and maintaining virtual machines. The core platform instances *NodeWebApp*, *DBServer*, *AuthServer*, and *FunctionalEngineServer*, together with supporting *Common Services* were created and synced with the *Testing* deployment.
- **AWS S3 (Simple Storage Service)** - The object storage service has been employed for three distinct purposes: (1) Storage back-end for SilicoFCM's back-end file manager, (2) Permanent storage for *FunctionalEngineServer* to store workflows' inputs and outputs, and (3) Temporary storage used during the execution of the workflows.
- **AWS Route 53** - DNS service for domain name resolution.

However, there are two major differences if we compare on-premise deployment to AWS deployment. The first one is the usage of S3 object storage to provide almost "infinite" storage space for *DBServer* and a collocated Girder back-end file manager. Thanks to the incorporation of Girder API into *FunctionalEngineServer*, the workflow engine is capable of seamlessly handling files, using them either as workflow inputs or workflow outputs.

The second difference from the on-premise deployment is heavy usage of the cloud features by *FunctionalEngineServer* thanks to the clustering and resource provisioning features provided by the back-end TOIL[7] workflow manager. *FunctionalEngineServer* now directly interacts with AWS EC2 API, launching and terminating virtual instances according to the current and provisioned workload, as depicted in Figure 3.

A. Workflow management in AWS environment

For compatibility purposes and to stay on track with modern trends and portability requirements, the SilicoFCM platform opted for CWL to provide all the workflows, including genomics, but also single-scale and multi-scale mechanics, CFD, electrodynamics, post-processing and many others. A special API entitled Functional Engine Server (FES-API) has been developed within SilicoFCM, capable of handling the entire life-cycle of multiple workflows simultaneously.

FES-API's responsibility is management of the workflows' life-cycle, including creation, execution, handling inputs and outputs, etc. FES-API is capable of using official CWL execution engine `cwltool` as an execution back-end, but also TOIL workflow executor. The latter has better capabilities to interface with AWS API directly, especially S3 and EC2 and

[5]https://www.nginx.com/

[6]https://girder.readthedocs.io/

[7]https://toil.readthedocs.io

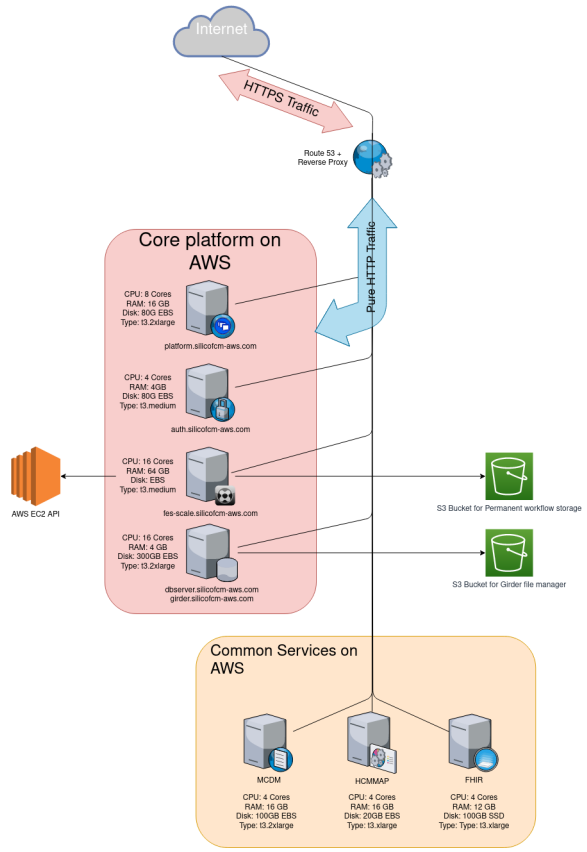

Fig. 3. SilicoFCM's deployment on AWS

to handle resource requests automatically. The architecture of the TOIL engine is shown in Figure 4. The only instance to be manually deployed is so-called *Leader*. In the auto-scale mode, the Node provisioner launches an appropriate EC2 instance if necessary, and joins it to the automatically managed Apache Mesos[8] cluster. As soon as an instance joins cluster, it is ready to execute a workflow requested by FES-API.

The scalability is not only provided when the load is increasing. One of the useful features of TOIL is that a provisioned EC2 instance automatically switches off upon completing the requested workflow and transferring the results to the appropriate S3 bucket. This feature contributes to infrastructure cost savings significantly, resolving the problem with "dangling" resources.

IV. Results and Discussion

Since the scaling policy of the Mesos cluster is fully dynamic, one has to count on certain provisioning overhead. EC2 launch penalty has to be added to each workflow's execution time. The alternative is to maintain a pool of ready EC2 instances all the time. However, if the overhead is acceptable, it is better to keep full auto-scaling on, for the sake of cost savings.

[8]http://mesos.apache.org/

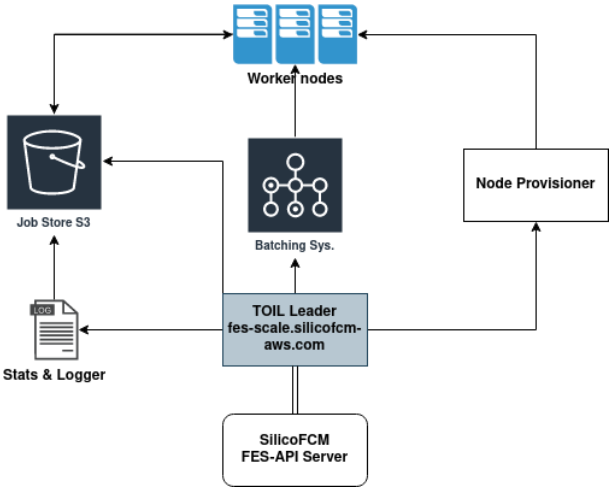

Fig. 4. The architecture of TOIL workflow manager in the AWS environment

In the local deployment, on each users' request, the workflow executes on an already up and running virtual machine. Contrary to that, on AWS deployment, a new VM launches whenever a user starts a workflow. We opted for `t3.2xlarge` instances, which are similar to our local cluster nodes in terms of computing power.

A. Performance comparison between local and cloud deployments

To benchmark the overhead, we took three representative workflows which significantly differ in execution time and required computing resources:

1) **O'Hara** - O'Hara model provides Calcium concentration given the parameters such as rate constant for dephosphorylation, fraction of active binding sites with a Ca^{2+}/calmodulin complex on them, rate constant for calmodulin complex binding to $CaMK$, as explained in [6] and [3].
2) **Pak-fs-patient** - Fluid-structure simulation on patient specific geometry Based on experimental data and DICOM files obtained from CT scanner. The realistic heart model consists from 35948 tetrahedral 3D elements, divided by 35948 nodes. The methodology is described in [10], [7], and [11].
3) **Torso-cwl** - Finite element model for ventricular activation sequence from ECG measurement. Determination of ventricular activation sequence from clinical ECG measurement using detailed heart and torso model of electrical field. More detail can be found in [8] and [9].

TABLE I
Execution times of certain workflows on local deployment compared to AWS deployment

	Local (s)	**AWS (s)**
O'Hara	123	381
Pak-fs-patient	497	612
Torso-cwl	4344	4896

Looking at O'Hara workflow in Table I and Figure 5, one can see that VM launch overhead on AWS takes up to 3 minutes, which is a significant amount for such a short workflow. However, the relative effect of this overhead is fading away for the long-running workflows such as *Pak-fs-patient* and *Torso-cwl*.

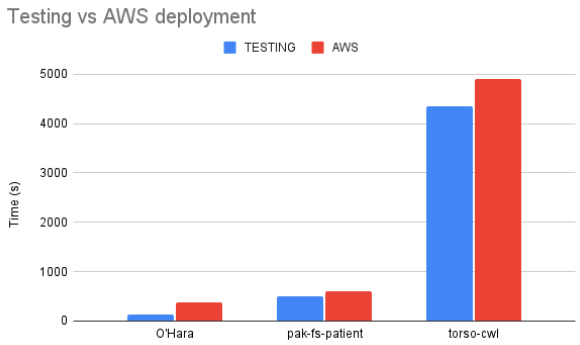

Fig. 5. The architecture of TOIL workflow manager in the AWS environment

B. Scalability and stress testing

The primary aim of the scalability feature in the SilicoFCM platform deployed on AWS is to provide enough resources under the heavy workload posed by the number of users running multiple workflows simultaneously. In order to test the behavior of the underlying AWS infrastructure effectively, we pursued a similar benchmark as the previous one but multiplied by a factor of 5. The main idea was to launch 5 instances of each kind of workflow simultaneously, measure execution time and variance, and compare them with the counterparts shown in Figure 5. The user launches the workflows using SilicoFCM web user interface, in order to mimic the real use case, i.e. performing a parameter study.

According to the benchmark results shown in Table II and Figure 6, there is a considerable variance column-wise for all three workflows. It is most visible in a short running workflow, such as *O'Hara*. If we look at *O'Hara* run number 3, it is much shorter than the others. The most likely cause for this result is the immediate availability of a proper cluster node in the Mesos cluster, without the need to launch a new EC2 node. Thus the launch overhead is negligible in run number 3.

TABLE II
SCALABILITY AND STRESS TEST - RUNNING MULTIPLE WORKFLOWS IN PARALLEL ON AWS

Execution no.	O'Hara	Pak-fs-patient (s)	Torso-cwl (s)
1	302	781	5764
2	359	658	8343
3	186	662	8281
4	295	597	8276
5	304	650	7794

While launch overheads are significant for the short-running workflows, for the long-running ones, such as *Torso-cwl*,

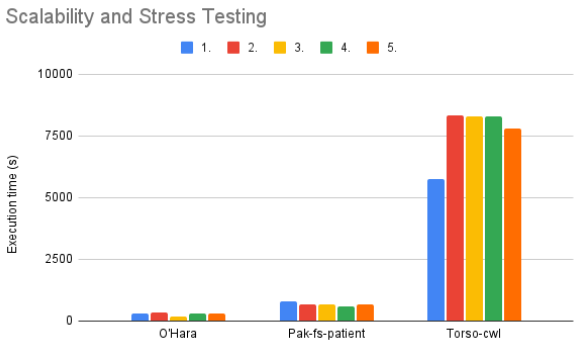

Fig. 6. Scalability and stress test - running multiple workflows in parallel

the most visible influence is that of CPU speed and the possible effect of over-provisioning and other similar effects characteristic for EC2 environment. Comparing Figs. 5 and 6, it is obvious that the times when the workflows run in parallel are from 18% to 70% longer than the situation when running only a single workflow. The obvious explanation is that underlying infrastructure (IaaS) is certainly non-linear in terms of performance metrics. EC2 service launches virtual instances sometimes on different physical servers, but sometimes a number of instances share the same physical server. In those situations, the instances also share resources such as I/O, system bus, etc. That is why one cannot expect the system to behave the same under multiplied load.

The workflow engine ported to AWS showed good performance when put to stress. The performance numbers are relatively stable and predictable. We can conclude that the SilicoFCM platform deployed on AWS is capable of sustaining sufficient performance under stress.

V. CONCLUSIONS

The primary aim of the cloudification of the SilicoFCM platform was to prove portability, improve scalability and reduce long-term infrastructure costs. The first objective is completely met, including certain added values. According to the demonstrated results, the second objective, scalability improvement, is also met. The system behaves predictive under moderate load, but also when put under stress. There is a certain launch overhead, which is relatively significant with computationally moderate workflows. However, the infrastructure launching overhead is almost negligible with any considerably computationally complex use-case. We have not yet carried out any study to analyze long-term infrastructure costs on AWS and compare them to the on-premise deployments, but it is planned for the future. The overall experience shows that a very complex multi-vendor and multi-component research platform can be ported to a public cloud without sacrificing the degree of control and performance, paving the way to similar undertakings in the future.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 777204 (SILICOFCM project – www.silicofcm.eu). This article reflects only the author's view. The Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] Enis Afgan, Dannon Baker, Marius Van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, et al. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research*, 44(W1):W3–W10, 2016.

[2] Allison P Heath, Matthew Greenway, Raymond Powell, Jonathan Spring, Rafael Suarez, David Hanley, Chai Bandlamudi, Megan E McNerney, Kevin P White, and Robert L Grossman. Bionimbus: a cloud for managing, analyzing and sharing large genomics datasets. *Journal of the American Medical Informatics Association*, 21(6):969–975, 2014.

[3] Milos Kojic, Miljan Milosevic, Vladimir Simic, Vladimir Geroski, Arturas Ziemys, Nenad Filipovic, and Mauro Ferrari. Smeared multiscale finite element model for electrophysiology and ionic transport in biological tissue. *Computers in biology and medicine*, 108:288–304, 2019.

[4] Bo Liu, Ravi K Madduri, Borja Sotomayor, Kyle Chard, Lukasz Lacinski, Utpal J Dave, Jianqiang Li, Chunchen Liu, and Ian T Foster. Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses. *Journal of biomedical informatics*, 49:119–133, 2014.

[5] Vivek Navale and Philip E Bourne. Cloud computing applications for biomedical science: A perspective. *PLoS computational biology*, 14(6):e1006144, 2018.

[6] Thomas O'Hara, László Virág, András Varró, and Yoram Rudy. Simulation of the undiseased human cardiac ventricular action potential: model formulation and experimental validation. *PLoS computational biology*, 7(5):e1002061, 2011.

[7] A van Oosterom. Source models in inverse electrocardiography. 2003.

[8] Alfonso Santiago. Fluid-electro-mechanical model of the human heart for supercomputers. 2018.

[9] Gerhard Sommer, Andreas J Schriefl, Michaela Andrä, Michael Sacherer, Christian Viertler, Heimo Wolinski, and Gerhard A Holzapfel. Biomechanical properties and microstructure of human ventricular myocardium. *Acta biomaterialia*, 24:172–192, 2015.

[10] A van Oosterom. The spatial covariance used in computing the pericardial potential distribution. *Computational Inverse Problems in Electrocardiography*, pages 1–50, 2001.

[11] A Van Oosterom. The equivalent double layer: source models for repolarization. In *Comprehensive Electrocardiology*. 2009.