Serverless Implementations of Real-time Embarrassingly Parallel Problems

Dimitar Mileski
Innovation Dooel
Skopje, North Macedonia

Marjan Gusev
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University in Skopje, North Macedonia

***Abstract*—In this paper, we conduct experiments to deploy a scalable serverless computing solution for real-time monitoring of thousands of patients with streaming electrocardiograms as an example of embarrassingly parallel tasks originally executed on two virtual machines. The research question is to find the speedup of such solution versus classical virtual machine approaches with sequential or parallel threads.**

The challenge of migrating an existing service to a serverless solution is to adapt and reconfigure the code for serverless platform, to write the code to invoke the service in parallel and asynchronously, and to use other services in the cloud that are needed for the whole solution to be functional and scalable. Evaluation of developing various solutions matching migration challenges to Google Cloud Run, Google Cloud Compute Engine, and Google Cloud Storage (customization of code, the configuration of services) shows that greater speedups can be achieved by dividing the Embarrassingly Parallel tasks into sub-tasks executed as a serverless service. We achieved highest speedup of almost 40 for Serverless solution compared to a sequential execution on a virtual machine solution, and speedup of 23 for Serverless solution compared to a Parallel execution using virtual machines.

***Index Terms*—serverless, containers, parallel, async, cloud migration, public cloud, cloud storage, cloud computing**

I. Introduction

Our initial solution of a real-time remote heart monitoring center was designed as a classical web application hosted on two virtual machines in a cloud [1]. We were challenged to realize several experiments aiming at an optimal computing environment that can provide real-time monitoring for thousands of simultaneous electrocardiogram (ECG) streams within the CardioHPC project [2]. In addition to the requirement to process thousands of patients simultaneously, the management insisted on a scalable platform providing sufficient quality (short response time) without worrying about the resources.

This paper describes the migration process of a classical cloud service to existing serverless services offered by the public clouds. The final goal is to achieve a more significant speedup for smaller costs.

Software migration is the process of switching from one operating environment to another that is considered better for software execution [3]. One type of software migration is cloud migration. A set of migration activities to support an end-to-end cloud migration is needed for the cloud migration process. Cloud migration defines a comprehensive perspective, capturing business, technical and scientific concerns with stakeholders with different backgrounds [4].

The higher speedup can be achieved by exploiting parallelism, and we classify our motivation task as an embarrassingly parallel computation with huge potential to run concurrently thousands of threads. Serverless is a platform that offers a scalable platform without worrying about resources. We specify 3 processing architecture types VM_Seq as a sequential solution on a classical Virtual Machine (VM) architecture, $VM\_Parallel$ executing parallel threads on a VM, and $Serverless\_Parallel$ solution as testing environments.

Our research question was to find the speedup achieved by the serverless and $VM\_Parallel$ compared to the VM_Seq solution. In addition, we discuss challenges solved to migrating an Embarrassingly Parallel Problem or a Nearly Embarrassingly Parallel Problem to a parallel serverless implementation.

The paper follows the next structure. Related work and analysis of the state-of-the-art are presented in Section II. System architecture, Experiments, and Evaluation methodology are described in Section III, illustrating the technical solution and approach that will achieve system scalability and how system speed and speedup will be measured. Results are evaluated and discussed in Section IV. Finally, Section V presents the Conclusions and future work directions.

II. Related Work

The idea of dividing a big computation into multiple independent tasks that can be invoked concurrently and executed on serverless infrastructure was exploited by Bharti et al. [5], proving that serverless platforms can be used cost-effectively for large-scale parallel processing applications. They propose a new approach to parallel serverless computing that has the benefit of successfully terminating the execution of AWS Lambda that will otherwise fail. The limiting factor of this approach was found to be that it is not applicable to problems that can not be divided into multiple independent tasks.

Nazari et al. [6] presented a literature review and state of the art on the serverless, pointing out how developers can run thousands of function instances concurrently for embarrassingly parallel jobs and gives a review of some limitations of serverless processing of embarrassingly parallel jobs. The research has shown that embarrassingly parallel problems are a good fit for serverless computing. A new way of developing data analytics applications with PyWren [7] on top of Serverless infrastructure gives benefits to PySpark [8],

This is accepted version. Published version DOI: 10.1109/TELFOR56187.2022.9983710

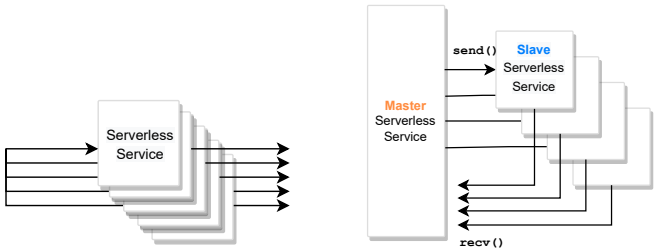

Fig. 1. Embarrassingly parallel (left) and Nearly embarrassingly parallel processing (right)

Hadoop [9] and other server-based distributed data analytic systems [6].

Ichnowski et al. [10] use multi-core serverless lambda computing for motion planning algorithms, exploiting both multi-core parallelism within the lambda functions and scalability of lambdas running in parallel, and conclude that serverless computing can effectively scale. Regarding the multi-core parallelism of serverless services, the performance of a parallelized function is limited by the allocated vCPUs, and the number of CPU cores available to the function/container does not always equal the number of allocated vCPUs [11]. That is a limiting factor in how much we can parallelize individual tasks inside embarrassingly parallel problems.

III. Methods and Technical Solutions

"Ideal" computational task can be divided into several completely independent parts to be executed by a separate processor (serverless service), a computational paradigm known as an embarrassingly parallel computation [12]. A truly embarrassingly parallel computation suggests no communication between the separate processes and hence requires no special techniques or algorithms to obtain a working solution. The left part of Fig. 1 shows truly embarrassingly parallel computation.

This paper focuses on serverless services instead of process or processor. An embarrassingly parallel computation requires no or very little communication [13]. Serverless services can execute the specified tasks without any interaction with other serverless services. In this ideal case, each serverless service receives independent raw data and generates results from these inputs without waiting for results from other serverless services.

Nearly embarrassingly parallel computations are those architectures that require intermediate calculations and results to be distributed, collected, and combined in some way [14]. Single serverless service must operate alone initially and finally. For smooth operation of embarrassingly parallel computation, a master serverless service is required in addition to slave serverless services (the right part of Fig. 1). The master serverless service is used for controlling the computational sequence, such as distributing the tasks among the slave serverless services and waiting for a process to be completed before assigning subsequent tasks. The available slave service instances conduct the computationally intensive tasks [14]. The computational load for each of the slave serverless service instances is about the same. Nearly embarrassingly parallel computation is achieved.

The master serverless service can use a static or a dynamic task assignment [13]. In static task assignment, each serverless service instance does a fixed part of the problem, known as a priori. Moreover, with a dynamic task assignment, a work-pool is maintained that serverless service instances consult to get more work [13]. The work pool is a collection of tasks to be executed. Serverless service instances execute new tasks as soon as they finish previously assigned tasks.

A. Migration challenges

The existing service to be migrated to a serverless implementation is presented in Fig. 2. Two Windows Server virtual machines are used in the existing solution as VMWare Cloud instances [15]. For this research, we focussed on the Beat Detection and Classification (BDC) service [16] as a compute-intensive module and realized experiments to migrate it to a serverless implementation. This service is invoked with an ECG file and runs signal processing algorithms that output annotations of detected and classified heartbeats. An ECG file to be processed is uploaded to a Web application deployed on a Windows Server 2012 R2. The web application is developed in .NET Core and Javascript. The Web application initiates the processing of the ECG file by calling the BDC Tomcat Web Server [17]. The BDC service processes the attached file, uses additional information (customized thresholds), and returns a processing status. The web application reads the processed files if the processing is completed successfully. Otherwise, it displays an error message that the BDC processing did not complete successfully.

Our earlier experiments showed that serverless Google Cloud offers more concurrent resources than Amazon Web Services [18], [19], and in this paper, we present the development of a serverless version of the BDC service on Google Compute Engine (GCE), Google Cloud Run (GCR), and Google Cloud Storage (GCS). GCE is an infrastructure service provided as a part of the Google Cloud Platform, providing three major components: virtual machines, persistent disks, and networks [20]. GCR is a fully managed computing environment to deploy and scale serverless HTTP containers without worrying about provisioning machines, configuring clusters, or autoscaling [21]. To implement the BDC service on GCR, we created a Docker [22] image of the BDC service and hosted it on Google Container Registry [23]. Corresponding ECG files are stored on GCS [24].

B. Solution architecture

The BDC serverless solution is shown in Fig. 3. It includes 3 services from Google GCE, GCR, and GCS. The serverless solution is an implementation of the Nearly Embarrassingly Parallel Problem, which means that in the solution, we will have one master serverless service and many slave serverless service instances.

The ECG file is uploaded to the web application. The web application in Fig. 2 is the same application in Fig. 3. The

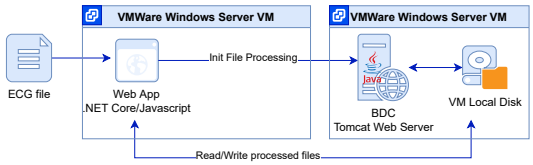

Fig. 2. BDC VM service architecture in cloud

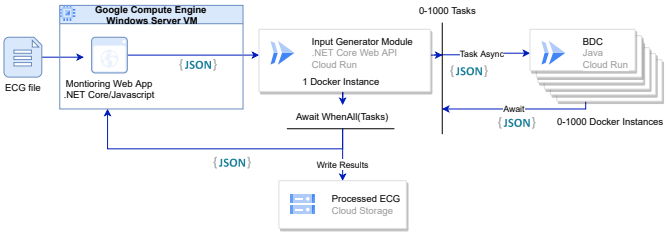

Fig. 3. BDC serverless service architecture in cloud

web application is deployed in a Google Compute Engine VM. The virtual machine has Windows Server 2022 operating system installed. All requests in the new solution work with JSON data-interchange format. The Input Generator Module IGM) represents a master serverless service and, BDC is a slave serverless service. The IGM is a .NET Core Web API. A docker image was created from the IGM and is pushed to Google Container Registry. BDC is a Java service with the same code as BDC from Fig. 2. A docker image was created from BDC that is pushed to Google Container Registry. The detailed configuration of the IGM and BDC in GCR is given in Table. I.

The IGM receives the data from the ECG file and divides it into equal time measurements of the ECG signal. InputFileChunks is a list of ECG signal chunks created by dividing the initial ECG file attached to the web application into equal time chunks. Then for each ECG signal chunk, BDC is called asynchronously.

Waiting for all asynchronous results from the BDC Slave service is synchronized using Task.WhenAll(tasks). That creates a task that will be finish when all of the supplied tasks have been completed. The results are collected from all BDCAPI results. The results are then written to a GCS bucket, and a JSON with processing status is returned.

```
var tasks = inputFileChunks.Select(async file => {
    BDCAPIResult = await callBDC(param);  //public
    static async Task<BDCModuleOutputParam> callBDC(
    param)
    File.APIResult = BDCAPIResult;
    return file;
});
foreach (var file in await Task.WhenAll(tasks)){
    collectResults(file);
}
```

Listing 1. IGM C# code

C. Experiments

The real-time scenario is based on processing the 30s or 60s ECG signal expecting a near real-time response (response time $\leq$ 3s).

There are 3 types of processingVM_Seq, $VM\_Parallel$, and $Serverless\_Parallel$. VM_Seq and $VM\_Parallel$ are executed in the BDC VM service architecture shown in Fig. 2. A $Serverless\_Parallel$ is executed in BDC serverless service architecture shown in Fig. 3.

TABLE I
CONFIGURATION OF SERVERLESS SERVICES ON GOOGLE CLOUD RUN

	IGM	BDC
Memory	4Gib	512MiB
vCPUs	8	1
Max requests per container	1	1
Min number of container instances	0	0
Max number of container instances	1	1000

D. Evaluation methodology

Two experiment test cases are specified with various execution configurations for durations of ECG files in minutes D and workloads of simultaneous W ECG streaming files, with the same total time of 500 minutes ECG measurement:

- Test case 1: where $D = 0.5$ and $W = 1000$,
- Test case 2: where $D = 1$ and $W = 500$.

The total execution time T_{total} is measured for each experiment, as average response time (latency) for 5 executions of the experiment.

To evaluate the impact of introducing the serverless parallelization, for each experiment we measure the response time $T_S(W)$ and $T_P(W)$, respectively for the VM sequential and serverless parallel version for a specific load W, and calculate the speedup $S(W)$ of the serverless service implementation with respect to the sequential VM version of the service, calculated by (1).

$$S(W) = \frac{T_S(W)}{T_P(W)} \tag{1}$$

IV. RESULTS AND DISCUSSION

Fig. 4 presents the achieved results of the total response time T_{total} and speedup $S(W)$ for a given configuration and different processing types. ($VMSeq\_VMParallel$, $VMParallel\_Serverless$ and, $VMSeq\_Serverless$) T_{total} is represented by a box and whisker plot where each T_{total} is a set of 5 executions of the experiment. Fig. 5 presents the speedup $S(W)$ obtained due to the introduction of parallelization.

Sequential processing in the VM has the largest T_{total}. We observe that the configuration with D=1 W=500 in all types of processing outperforms D=0.5 W=1000. It is better to have 500 ECG files of 1 minute measurement rather than 1000 files of 30 seconds. The lowest $S(W)$ is achieved for $VMSeq\_VMParallel$ and the highest for $VMSeq\_Serverless$. In all three types of real-time scenario processing, the most significant speedup $S(W)$ is for W=0.5 D=1000. Although for the experiment D=0.5 W=1000, we achieve a worse T_{total} for all processing types, still more W

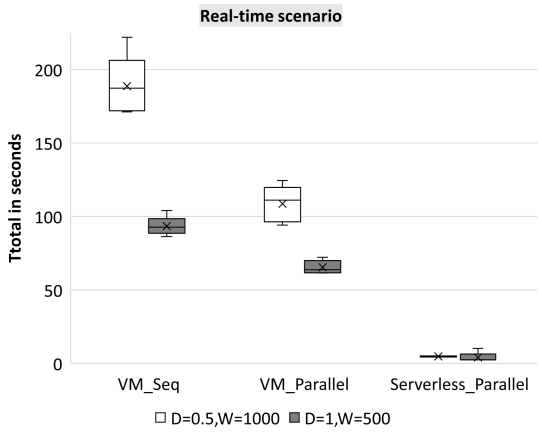

Fig. 4. Total execution time for each experiment (latency)

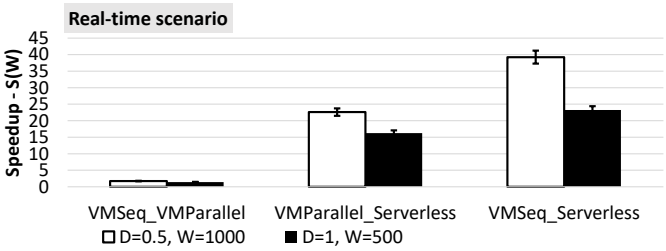

Fig. 5. Speedup

will cause more service requests, which means more serverless instances will process in parallel. The highest speedup $S(W)$ of almost 40 is achieved for $VMSeq\_Serverless$, and 23 for $VMParallel\_Serverless$.

V. Conclusion

We conducted experiments for three processing architecture types VM_Seq, $VM\_Parallel$ and, $Serverless\_Parallel$. The evaluation confirmed maximum speedup of $S(W) = 40$ conducting the real-time experiments. Better speedup for real-time requires larger W with smaller D of ECG signal to spin up a greater number of instances that will process in parallel.

Serverless computing model can bring significant speedup for Embarrassingly parallel problems and Nearly Embarrassingly parallel problems. The migration challenges are: dividing the problem into sub-problems, fast and parallel sending of requests to serverless platforms that will spin up a large number of instances to process in parallel, incorporating other services such as cloud storage for the whole solution to be functional and elastic.

Acknowledgment

The experiment "CardioHPC - Improving DL-based Arrhythmia Classification Algorithm and Simulation of Real-Time Heart Monitoring of Thousands of Patients" has received funding from the European High-Performance Computing Joint Undertaking (JU) through the FF4EuroHPC project under grant agreement No 951745. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Italy, Slovenia, France, and Spain.

References

[1] M. Gusev, A. Stojmenski, and A. Guseva, "Ecgalert: A heart attack alerting system," 09 2017, pp. 27–36.
[2] Innovation-Dooel, "Cardiohpc - real-time heart monitoring of thousands of patients." [Online]. Available: https://bit.ly/cardiohpc
[3] M. Semilof, K. Casey, and J. Montgomery, "What is cloud migration? an introduction to moving to the cloud," Dec 2021. [Online]. Available: https://www.techtarget.com/searchcloudcomputing/definition/cloud-migration
[4] C. Pahl, H. Xiong, and R. Walshe, "A comparison of on-premise to cloud migration approaches," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2013, pp. 212–226.
[5] U. Bharti, D. Bajaj, A. Goel, and S. Gupta, "A novel design approach exploiting data parallelism in serverless infrastructure," in *Advances in Computing and Network Communications*. Springer, 2021, pp. 247–260.
[6] M. Nazari, S. Goodarzy, E. Keller, E. Rozner, and S. Mishra, "Optimizing and extending serverless platforms: A survey," in *2021 Eighth International Conference on Software Defined Systems (SDS)*. IEEE, 2021, pp. 1–8.
[7] J. Sampé, G. Vernik, M. Sánchez-Artigas, and P. García-López, "Serverless data analytics in the IBM cloud," in *Proceedings of the 19th International Middleware Conference Industry*, 2018, pp. 1–8.
[8] T. Drabas and D. Lee, *Learning PySpark*. Packt Publishing Ltd, 2017.
[9] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.
[10] J. Ichnowski, W. Lee, V. Murta, S. Paradis, R. Alterovitz, J. E. Gonzalez, I. Stoica, and K. Goldberg, "Fog robotics algorithms for distributed motion planning using lambda serverless computing," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4232–4238.
[11] M. Kiener, M. Chadha, and M. Gerndt, "Towards demystifying intra-function parallelism in serverless computing," in *Proceedings of the Seventh International Workshop on Serverless Computing (WoSC7) 2021*, 2021, pp. 42–49.
[12] M. Allen, B. Wilkinson, and J. Alley, "Parallel programming for the millennium: Integration throughout the undergraduate curriculum," in *Second Forum on Parallel Computing Curricula*, 1997.
[13] J.-C. Régin, M. Rezgui, and A. Malapert, "Embarrassingly parallel search," in *International conference on principles and practice of constraint programming*. Springer, 2013, pp. 596–610.
[14] N. Wang, Y.-Z. Chang, and C.-M. Tsai, "The application of nearly embarrassingly parallel computation in the optimization of fluid-film lubrication©," *Tribology transactions*, vol. 47, no. 1, pp. 34–42, 2004.
[15] S. Lowe, *Mastering vmware vsphere 5*. John Wiley & Sons, 2011.
[16] E. Domazet and M. Gusev, "Improving the QRS detection for one-channel ECG sensor," *Technology and Health Care*, vol. 27, no. 6, pp. 623–642, 2019.
[17] J. Brittain and I. F. Darwin, *Tomcat: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2007.
[18] D. Mileski and M. Gusev, "Serverless FaaS scalability evaluation: An ECG signal processing use case," in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2022, pp. 853–858.
[19] M. Gusev, S. Ristov, A. Amza, A. Hohenegger, R. Prodan, D. Mileski, P. Gushev, and G. Temelkov, "CardioHPC: Serverless approaches for real-time heart monitoring of thousands of patients," in *WORKS 22, 17th Workshop on Workflows in Support of Large-Scale Science, Super Computing, SC22 Conference*, 2022.
[20] S. Krishnan and J. L. U. Gonzalez, "Google compute engine," in *Building your next big thing with Google cloud platform*. Springer, 2015, pp. 53–81.
[21] E. Bisong, "An overview of google cloud platform services," *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 7–10, 2019.
[22] D. Merkel *et al.*, "Docker: lightweight linux containers for consistent development and deployment," *Linux j*, vol. 239, no. 2, p. 2, 2014.
[23] Google, "Container registry documentation; google cloud." [Online]. Available: https://cloud.google.com/container-registry/docs
[24] G. C. S. Documentation, "Cloud storage documentation; google cloud." [Online]. Available: https://cloud.google.com/storage/docs