Serverless Computing: From Planet Mars to the Cloud

José Luis Vázquez-Poletti
Universidad Complutense de Madrid

Ignacio Martín Llorente
Universidad Complutense de Madrid
Harvard University

Editors:
Konrad Hinsen;
konrad.hinsen@cnrs.fr,

Matthew Turk;
matthewturk@gmail.com

Serverless computing is a new way of managing computations in the cloud. We show how it can be put to work for scientific data analysis. For this, we detail our serverless architecture for an application analyzing data from one of the instruments onboard the ESA Mars Express orbiter, and then, we compare it with a traditional server solution.

SERVERLESS AND (PUBLIC) CLOUD COMPUTING

Serverless computing is an execution model where the user provides a code to be run without any involvement in server management or capacity planning. Many serverless implementations are offered in the form of compute runtimes, which execute application logic but do not persistently store data. As the uploaded code is exposed to the outside world, the developer no longer needs to be involved in multitasking, handling requests, or operating system costs (installation, maintenance, and licenses). Serverless computing is a form of utility computing and paradoxically relies on actual servers maintained by cloud computing providers.

Cloud computing is a provision model that gives access to dynamic, elastic, and on-demand computational resources. Within the general deployment models, public clouds have attracted much interest among the scientific community in recent years. Public cloud services are provided by an independent organization that owns compute resources, which are then offered to their customers. Public cloud users, especially those from the science field, find this pay-as-you-go pricing model to be very convenient. Its great flexibility allows budgets to be easily adapted to computing needs.

Published by the IEEE Computer Society
1521-9615/18/$33.00 ©2018 IEEE

Amazon web services (AWS) is one of the most widely used public cloud platforms. Among their numerous services, Lambda (aws.amazon.com/lambda) offers what can be regarded as serverless computing. Under the premise "run code, not servers," AWS Lambda makes it possible to run code in response to specific events while transparently managing the underlying compute resources for the user.

AN APPLICATION FROM A NEIGHBORING PLANET

The application ported to the Lambda serverless environment processes data from the Mars Advanced Radar for Subsurface and Ionosphere Sounding (MARSIS).[1] MARSIS is a pulse-limited and low-frequency radar sounder and altimeter installed on the Mars Express orbiter from the European Space Agency (sci.esa.int/mars-express/), which has been traveling around Mars since 2003. The instrument gained fame in July 2018 when it detected liquid water hidden beneath the Martian South Pole.[2]

The application processes MARSIS data from the active ionospheric sounding (AIS) experiment and displays it graphically in order to identify magnetic fields. It also allows for the detection of induced magnetic fields deep in the Martian ionosphere, compressing its plasma.[3] Through this process, new avenues have become available to study the effects of solar wind and understand dust storms.

Figure 1 shows the process performed by the application, which reuses software previously used in the mission.

Our objective was to process every data file once it became available. This minimized response time, achieving an optimal performance-cost trade-off. The Mars Express mission should be extended to at least the year 2022, and the study of the ionosphere has been identified as one of the mission's priorities, as observations performed by the orbiter will see their coverage augmented and their long-time series extended.

In light of this, a large MARSIS dataset was generated as our starting point to evaluate the best computing approach. This consisted of 9761 files with a total size of 92 GB and was retrieved between 2005 and 2016 (see Figure 2). The idea was that if our proposal worked with this large dataset, it should also be suitable for future data.

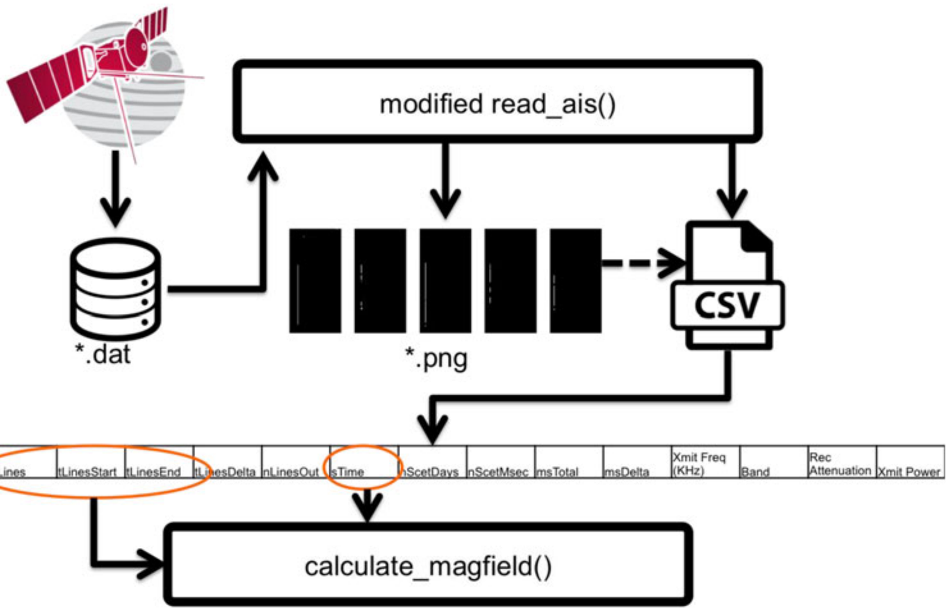

Figure 1. Application overview. Data from the AIS experiment are converted into images, which are then used to calculate magnetic fields.

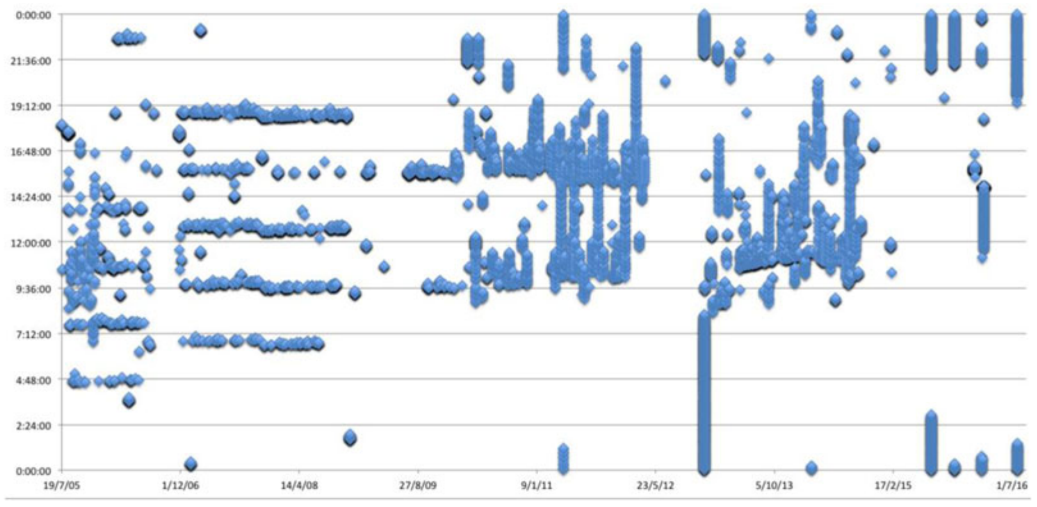

Figure 2. Arrival dates and times of the entire dataset from the AIS experiment between 2005 and 2016.

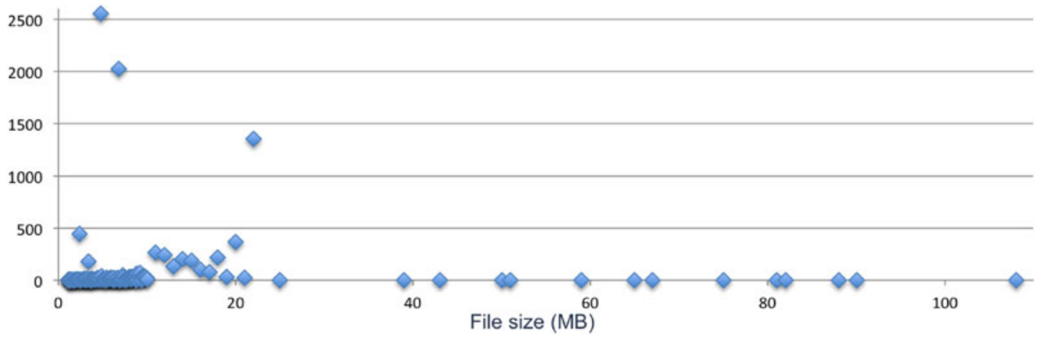

Figure 3. Distribution of data file sizes from the AIS experiment between 2005 and 2016 (in MB).

Execution time may take from milliseconds to half a minute for each data file depending on its size (see Figure 3). As there were many independent, simple processes involved, where each computation needed to be optimized for latency and completed in near real time, we considered porting to a serverless environment to be the next logical step.

GETTING IT DONE WITH AWS LAMBDA

With regard to Lambda, a so-called function was created that contained the following.

1. An executable that performs the core operations on the specified data file. This executable, programed in C, was generated on an Amazon Linux box with static libraries. Its size is 21 KB.
2. The main function code that handles the data file, invoking the executable and copying the output to a specified repository. Node.js 6.10 was used to program it, but Lambda also offers Java, C#, and Python bindings. Its size is 11.1 KB.

The main function code is also responsible for retrieving a reference file needed in the process. The reason for not including it in the function bundle is its size (77 MB), which exceeds the maximum allowed. All output files must be generated in a scratch directory (/tmp) limited to 512 MB. The rest of the filesystem where the function is executed is read-only.

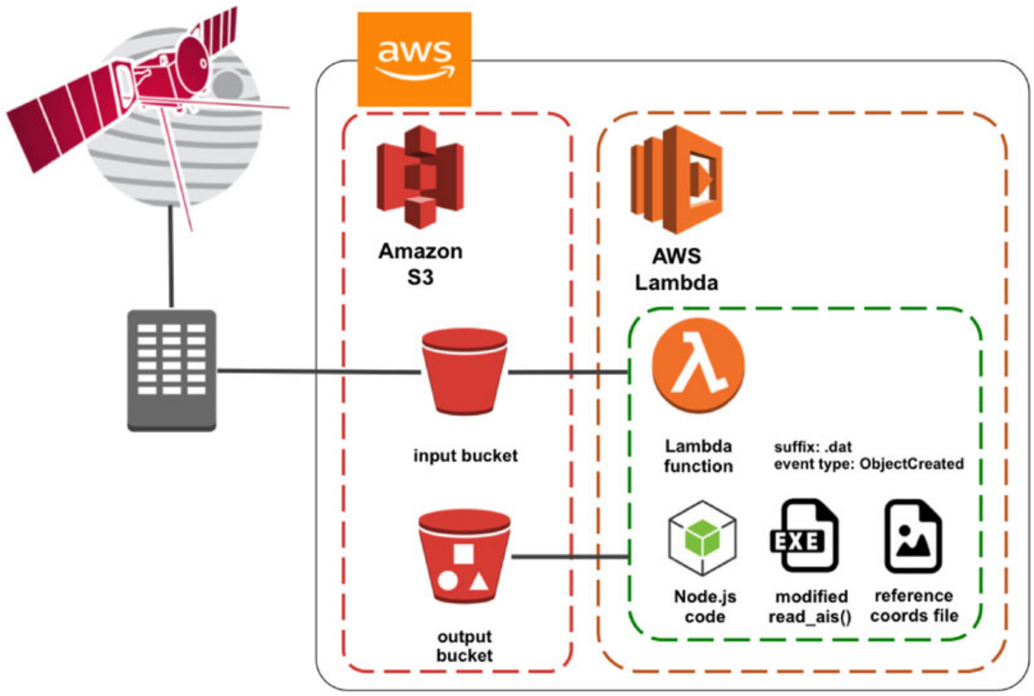

Figure 4. Architecture overview showing AWS services involved. Official icon set provided by Amazon web services.

The function was then configured to be triggered every time a new file with the .dat suffix appeared in a specified simple storage service (S3) bucket. In Lambda, there are many event sources that can be used for triggering functions, all of them related to AWS services. Some examples are email reception by simple email service, data addition to a DynamoDB table, and scheduled triggers using CloudWatch events.

Returning to our function, output files are moved to another S3 bucket. An overview of the architecture is shown in Figure 4.

Since Lambda is a serverless service, users do not need to be concerned with server specifications. Functions were conceived to be small and short, and the price depends on the following parameters.

- Memory assigned to the function. We specified 704 MB, with a maximum of 3008 MB.
- Deadline. Even if 30 s could be enough, we specified 1 min, with a maximum of 5.

When one of the previous limits is reached, Lambda stops the execution of the function.

TUNING OUR FUNCTION: COLD AND WARM CONTAINERS

When a function is invoked, Lambda releases a container (sandbox) to be allocated within the AWS infrastructure, function files are copied onto it, and execution takes place. If several trigger events occur at the same time, several containers are released in order to meet the demand.

The lifetime of each of these containers is 15 min after the execution of the function. If there is a new invocation, the container is reused, including all of its files. We took advantage of this in order to speed up the execution of the function. For instance, a warm (reused) container does not require the function to obtain the reference file (77 MB), because it has already been downloaded. Also, a warm container skips the Node.js language initialization and the code initialization itself.

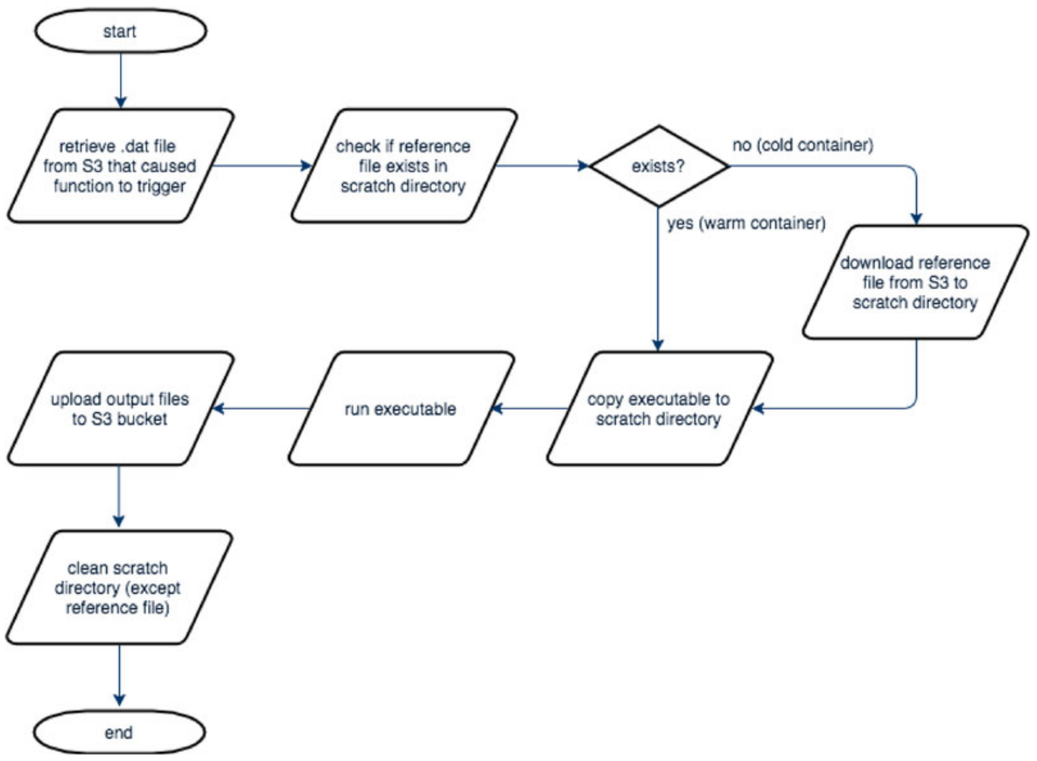
start
retrieve .dat file from S3 that caused function to trigger
check if reference file exists in scratch directory
exists?
no (cold container)
yes (warm container)
download reference file from S3 to scratch directory
copy executable to scratch directory
run executable
upload output files to S3 bucket
clean scratch directory (except reference file)
end

Figure 5. Flowchart describing the lifecycle of our AWS Lambda function.

We finally estimated the container average warming time (execution time in cold containers minus that in warm ones) to be 5 s.

The full function structure is described in Figure 5.

SERVERLESS OR SERVERFUL? A PERFORMANCE AND COST ANALYSIS

This section compares our serverless solution with a more traditional server-based one, but still in the cloud, using AWS Elastic Compute Cloud (EC2).

In the AWS EC2 case, a t2.small machine (1 virtual CPU, 2 GB memory) was prepared with everything needed for processing the input files when they became available, including the reference file. As keeping this machine up 24/7 was not a valid option, the following procedure was considered.

1. Start the machine and download the .dat file from the S3 input bucket.
2. Run the executable, one .dat file at a time.
3. Upload the output file to the S3 output bucket.
4. If there is a new .dat file in less than 1 min, go to 2.
5. Stop the machine.

In the EC2 simulations, we assumed the existence of an external program that detects the availability of new .dat files in the S3 input bucket. As explained before, this process is automatic in Lambda, where the user can easily define triggers.

Stopping the machine ensures that its configuration and files (executable, reference file) will be ready for the next process without any further cost (which is \$0.023/h when running). Starting the machine takes an average of 26 s, and the file transfer takes around 4 s.

The arrival of files, as detailed in Figure 2, was simulated in EC2. The machine would be up a total of 173 h 38′ (including the 1′ wait for new .dat files). The execution of 4776 processes (48.92% of the dataset) was delayed because of machines that were either stopped or busy. In the first case, boot time delays were applied. In the second, a new machine was started if its boot time was lower than the estimated remaining time for the active process.

During the simulations, no more than two simultaneous instances were needed. This happened 512 times. However, the accumulated delay was 39 h 20′: an average of 29 s for each of the 4776 affected processes.

Using the virtual machine approach costs \$3.99. Storage with S3 involves a very small cost, as it charges \$0.023/GB when transfers are less than 50 TB/month. The total cost would be \$2.21, \$2.12 corresponding to input transfers (92 GB) and \$0.092 to output transfers (4 GB).

Moving to Lambda, the total usage time was 10 h 11′, but AWS would bill 8 min more because of rounding up. With regard to container temperature, 898 were cold and 8863 were warm, taking advantage of the tweak described before. Also, 5 s was the average time for container warming, whereas the accumulated delay for Lambda was 1 h 15′.

Table 1 shows a comparison of the execution times in both solutions for a set of file sizes. As shown, Lambda offers the best performance for all selected cases.

Prices in Lambda depend on the chosen configuration. In our case (704 MB and 1′ deadline), the total cost would be \$4.25 just for execution. Storage has the same cost as in the EC2 solution (\$2.21), as S3 buckets are used.

As can be seen in the breakdown of costs for each of the solutions shown in Table 2, the EC2 solution is slightly cheaper than the Lambda one. On the other hand, Table 1 shows that execution time and delays for Lambda are lower. Moreover, execution on EC2 would require the development of a trigger system.

AWS Lambda can be a valid solution if the demand is for low latency processing at a reasonable cost, and the underlying infrastructure is not a concern (only two performance parameters can be specified). Serverless computing requires processes to be atomized, and Lambda is limited by the Amazon Linux image it comes with, static libraries, and what can be installed and used in less than the time available for execution. If you rely on some of the many services offered by Amazon or you are willing to

Table 1. Comparison of execution times in AWS EC2 and Lambda considering file sizes.

File size (MB)	Execution time AWS EC2 (ms)	Execution time AWS Lambda (ms)
4.9	3374	1800
13	9228	3700
17	11 796	4600

Table 2. Costs for each of the solutions.

	AWS EC2	AWS Lambda
Execution	\$3.99	\$4.25
Storage	\$2.21	

migrate from other providers, orchestration with Lambda is a must. Obviously, the drawback comes in the form of vendor lock-in.

ACKNOWLEDGMENTS

The authors would like to thank the ESA Mars Express mission team (in particular, its Project Scientist, Dr. Dmitri Titov) and their colleagues from the UCM Martian Studies Group (in particular, Prof. Luis Vázquez, Dr. María Ramírez-Nicolás, Dr. Pedro J. Pascual, Dr. Salvador Jiménez, and Dr. David Usero).

The authors are also grateful for the support provided by the Spanish Ministry of Economy and Competitiveness under project number TIN2015-65469-P and by the European Commission under the IN-TIME project under Grant 823934.

REFERENCES

1. G. Picardi, D. Biccardi, R. Seu, J. Plaut, W. T. K. Johnson, and R. L. Jordan, "MARSIS: Mars advanced radar for subsurface and ionosphere sounding," in *Mars Express: The Scientific Payload*. Noordwijk, The Netherlands: ESA Publ. Div., 2004, pp. 51–69.
2. R. Orosei *et al.*, "Radar evidence of subglacial liquid water on Mars," *Science*, 2018, http://science.sciencemag.org/content/early/2018/07/24/science.aar7268
3. M. Ramírez-Nicolás *et al.*, "The effect of the induced magnetic field on the electron density vertical profile of the Mars' ionosphere: A Mars Express MARSIS radar data analysis and interpretation, A case study," *Planet. Space Sci.*, vol. 20, pp. 49–62, 2016.

ABOUT THE AUTHORS

José Luis Vázquez-Poletti is an Associate Professor with the Universidad Complutense de Madrid, Madrid, Spain, where he is also the Head of its Open Source Software and Open Technologies Office. His research interests include distributed, parallel, and data-intensive computing technologies, as well as innovative applications of those technologies to the scientific area. He received the Ph.D. degree in computer architecture from the Universidad Complutense de Madrid. Contact him at jlvazquez@fdi.ucm.es.

Ignacio Martín Llorente is a Visiting Professor with the Harvard John A. Paulson School of Engineering and Applied Sciences, Cambridge MA, USA, a Professor in computer architecture and the Head of the Data-Intensive Cloud Lab, Universidad Complutense de Madrid, Madrid, Spain, and the Co-Founder and Director of the OpenNebula open-source project for cloud computing management. His research interests include distributed, parallel, and data-intensive computing technologies, as well as innovative applications of those technologies to business and scientific problems. Contact him at imllorente@ucm.es or imllorente@seas.harvard.edu.