The Astrophysical Journal Supplement Series, 240:32 (7pp), 2019 February https://doi.org/10.3847/1538-4365/aafcb5

A Cloud-based Architecture for the Cherenkov Telescope Array Observation Simulations: Optimization, Design, and Results

M. Landoni[1], P. Romano[1], S. Vercellone[1], J. Knödlseder[2], A. Bianco[1], F. Tavecchio[1], and A. Corina[3]
[1] INAF, Osservatorio Astronomico di Brera, Via E. Bianchi 46 I-23807 Merate (LC), Italy
[2] Institut de Recherche en Astrophysique et Planetologie, 9 avenue Colonel-Roche, F-31028, Toulouse Cedex 4, France
[3] Amazon Web Services Inc., P.O. Box 81226, Seattle, WA 98108-1226, USA
Received 2018 September 13; revised 2018 December 28; accepted 2018 December 31; published 2019 February 7

Abstract

Simulating and analyzing detailed observations of astrophysical sources for very high energy experiments, like the Cherenkov Telescope Array (CTA), can be a demanding task especially in terms of CPU consumption and required storage. In this context we propose an innovative cloud computing architecture based on Amazon Web Services (AWS) aiming to decrease the amount of time required to simulate and analyze a given field by distributing the workload and by exploiting the large computational power offered by AWS. We detail how the various services offered by the Amazon online platform are jointly used in our architecture, and we report a comparison of the execution times required for simulating observations of a test source with CTA, by a single machine and the cloud-based approach. We find that, using AWS, we can run our simulations more than 2 orders of magnitude faster than using a general purpose workstation for the same cost. We suggest considering this method when observations need to be simulated, analyzed, and concluded within short timescales.

Key words: methods: data analysis – methods: numerical – methods: statistical

1. Introduction

While the very high energy (VHE, above a few tens of GeV) γ-rays astrophysics community is obtaining astounding results with the analysis of data from the current generation of ground-based imaging atmospheric Cherenkov telescopes (IACTs, see Hinton & Hofmann 2009; Lemoine-Goumard 2015), a new generation of IACTs is already being developed. The Cherenkov Telescope Array (CTA) has been proposed (Actis et al. 2011; Acharya et al. 2013) to boost dramatically the current IACT performance and to increase the breadth and depth of VHE science.

To obtain the required wide energy range covered by CTA (from 20 GeV up to 300 TeV), the array will be composed of different classes of telescopes, namely, the large-sized telescopes (LSTs, $D \sim 23$ m), which will lower the energy threshold down to a few tens of GeV; medium-sized telescopes (MSTs, $D \sim 12$ m), which will improve the sensitivity in the 0.15–5 TeV energy range by a factor of 5–10; and small-sized telescopes (SSTs, primary mirror $D \sim 4$ m) from which the study of the Galactic plane in the energy range beyond 100 TeV will benefit the most. Furthermore, the full array will be installed in two sites, one for each hemisphere to allow an all-sky coverage. The baseline setup currently includes (Hofmann 2017a, 2017b) 4 LSTs and 15 MSTs in the northern site, covering an area of $\sim$1 km^2, at the Observatorio del Roque de los Muchachos on the island of La Palma (Spain); and 4 LSTs, 25 MSTs, and 70 SSTs in the southern site, covering an area of about 4 km^2, at the European Southern Observatory's Paranal Observatory in the Atacama Desert (Chile).

CTA is currently in the scientific assessment phase of simulating feasibility and scientific return of potential astrophysical targets that, in turn, can be used to determine future observing plans that maximize the overall payoff along the whole CTA lifetime. This often implies episodic, highly CPU-intensive simulations that are performed on specific science projects within broader topics on very short timescales. Under these conditions it is generally not cost effective to purchase, set up, and maintain a large enough cluster of computers to perform the task, and it may be cheaper to buy CPU time (and all correlated services of moving and storing large amounts of data) in a cloud platform (see e.g., Williams et al. 2018).

In order to use cloud computing two steps need to be taken. The first is to choose a cloud platform among the many currently available (e.g., Amazon Web Services (AWS), Google Cloud Platform), which offers the flexibility of a solution tailored to everyone's computing and storage requirements, which can also meet their financial constraints. The second step is to adapt all software that needs to be run for the simulations so they can run in a parallel fashion (i.e., able to exploit many cores on a machine) and to be distributed, thus sharing the workload among many computers across the network.

For the purpose of our simulations for CTA we chose AWS[4] thanks to a combination of high reliability, low cost, ease of service, and previous positive experience (Genoni et al. 2017). We also used the Docker platform[5] to ease the distribution of the tasks to run.

To perform the case study we present in this work we used `ctools` (Knödlseder et al. 2016, v. 1.4.2),[6] an analysis package for IACT data. `ctools` is not designed to be operated parallel and distributed across many different nodes. For this reason we ran each simulation as a sequence of `ctools` tasks, executed through `shell` scripts, independently in a single thread without a message passing between processes throughout the network. Final results are stored at the end of computation in the local file system of each node.

In this paper we show an example of an extensive set of simulations based on a Monte-Carlo sampling of the CTA

[4] https://aws.amazon.com/
[5] https://www.docker.com/
[6] http://cta.irap.omp.eu/ctools/

Table 1
Array of `ctools` Simulations

Model	IRF	Expo. (h)	Sim. N	CPU Time[a]
Crab	`South_z20_average_5h`	5	1000	$8^h\ 48^m$
0.1Crab	`South_z20_average_5h`	5	1000	$8^h\ 6^m$
0.01Crab	`South_z20_average_5h`	5	1000	$9^h\ 33^m$
mCrab[b]	`South_z20_average_5h`	5	1000	$24^h\ 58^m$
mCrab[b,c]	`South_z20_average_5h`	10	1000	$43^h\ 10^m$
mCrab	`South_z20_average_50h`	50	1000	$74^h\ 27^m$

Notes. All realizations were obtained considering only the southern CTA site and a zenith angle of 20°.
[a] Run time for 1000 realizations (single core).
[b] The source is not detected in most of the realizations (see Figure 4 and Table 2).
[c] We used the IRFs relative to the closest exposure.

Instrument Response functions of test astrophysical sources as seen through the eyes of the forthcoming CTA. We briefly describe the tasks executed to perform our simulations and the requirements for their parallelization and distribution (Section 2). We then describe our novel approach to running these simulations based on cloud computing (Section 3). Finally, the results and the advantages are discussed in terms of optimization of computing times using cloud computing for this kind of work in Sections 4 and 5.

2. A Case Study

As a case study we considered four test, point-like sources with a simple Crab-like power-law spectrum spanning 4 orders of magnitude in the flux (1, 0.1, 0.01, and 0.001 Crab). These sources can be used to estimate the run times for more realistic sources, as we discuss in Section 4. Our simulations were performed with the `ctools` package and the public CTA instrument response files[7] (IRF, v. prod3b-v1).

To define the spectrum of our test sources we adopted a power-law spectral model, described as a monochromatic flux,

$$M_{\rm spectral}(E) = k_0\left(\frac{E}{E_0}\right)^{\gamma}, \quad (1)$$

where k_0 is the normalization (or the prefactor, in units of ph cm^{-2} s^{-1} MeV^{-1}), E_0 is the pivot energy in MeV,[8] and γ is the power-law photon index. A Crab-like power-law spectrum is, therefore, described by $k_0 = 5.7 \times 10^{-16}$ ph cm^{-2} s^{-1} MeV^{-1} and $\gamma = -2.48$, so that in the 0.1–100 TeV band 1 Crab $= 5.248 \times 10^{-10}$ erg cm^{-2} s^{-1}. We placed our test sources at the coordinates of a known galaxy, NGC 1068, R.A.(J2000) $= 02^h\,42^m\,40^s.70$, decl.(J2000) $= -00°00'48''.0$, so that they are visible from both CTA sites at a zenith angle of 20°. For purposes of method validation and calculation of run times, we only consider the southern site because it provides a wider energy coverage. We, therefore, chose our IRFs based on coordinates and required exposure time, as reported in Table 1 column 2.

For the residual cosmic-ray background we included the instrumental background described in the IRFs, and no further contaminating astrophysical sources in the 5° field of view (FOV) were considered for event extraction.

We define a "simulation" as a set of N independent realizations. Each realization is performed through a `shell` script that drives a sequential series of commands, alternating purely astrophysical computations performed through `ctools` tasks, and housekeeping scripts. In our specific case, a realization includes first running the task `ctobssim` within `ctools` to create one event list based on our input model, including background events that were randomly drawn from the background model that is shipped with the IRFs. The randomization is controlled by a seed that is unique to this realization. Subsequently, the task `ctlike` reads in the event file and the input model file and, using an unbinned maximum likelihood model fitting, determines the best-fit spectral parameters from which we derive the flux, as well as the covariance matrices and the test statistics (TS) value (Mattox et al. 1996).

To overcome the impact of a given statistical realization on the fit results, we performed for each spectral model sets of $N = 1000$ as statistically independent realizations by changing the `ctobssim` seed value, thus calculating 1000 sets of each spectral parameter and TS. The value of each spectral parameter and its uncertainty are then calculated as the mean and standard deviation drawn from the distribution of the values of such a parameter.

In our case each simulation is described by a "simulation table," i.e., a list of $N = 1000$ calls to a script that will perform one realization. For the simulation table, in order to run in a parallel fashion, each step needs to be uniquely dependent on the randomization seed and to rely on uniquely defined variables and input/output files.

A summary of the simulations is shown in Table 1, where for each test source, column 2 reports the IRF chosen, column 3 reports the exposure, column 4 reports the number of realizations run for each simulation, and column 5 reports the typical run time for 1000 realizations. We used an Intel® Xeon® CPU E5-2620 v4 @ 2.10 GHz machine (eight cores) with 82 GB RAM, running RedHat 7.5 with gcc compiler v.4.8.5, for all simulations.

3. Living on a Cloud

The key to increase the performance of trivially parallel simulations by orders of magnitude where no sustained network communication between processes is required resides in the parallelization and distribution of the workload across a cluster of computers spread out through the network. This could be reached using an arbitrarily large number of computers with parallel use of CPUs on each node. In practice a compromise between the overall run time and cost (the faster, the more expensive) must be reached depending on the goals and time-frame of the project and its funding. In what follows we describe our solution, and we briefly review the main services used to achieve our goal.

3.1. The Docker Platform

The first step of the distribution of the simulations is to make a fully functional image of the software that can be duplicated indefinitely on any computer. We reached this goal with `Docker`,[9] a software program that allowed us to "containerize" all

[7] https://www.cta-observatory.org/science/cta-performance/
[8] Generally fixed at 10^6 MeV.
[9] https://www.docker.com/

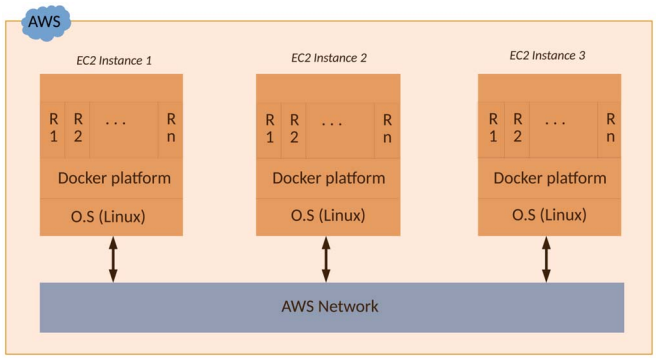

Figure 1. AWS EC2 method for the distribution of the workload: the `ctools` suite is containerized through a `Docker` in each instance. Each instance allows a large number of realizations to be run simultaneously.

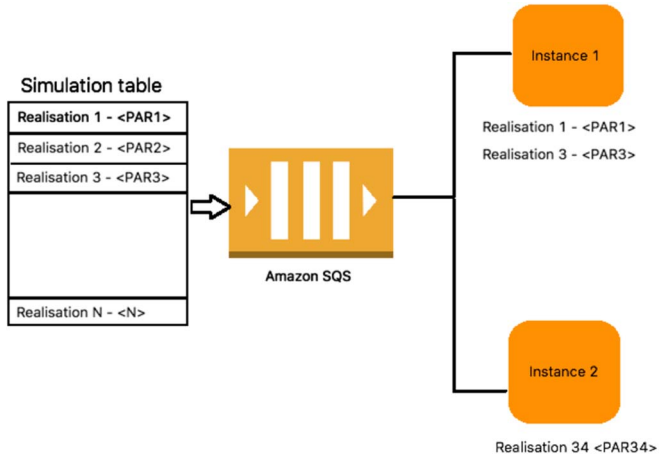

Figure 2. AWS SQS and AWS EC2 interfaces for the distribution of the workload. At the beginning the table is pushed to SQS and run time, where each instance receives some rows of realizations that are performed using the `ctools` suite containerized through `Docker`.

relevant software into one single virtual executable file (called Docker Image), including libraries, environment configurations, and software installation, which is fully portable from one computer to another in a complete platform-independent fashion. Furthermore, while `Docker` works like a virtual machine, it is nonetheless orders of magnitude lighter than the latter. This allows the simultaneous execution of dozen of runs, called "containers," of the executable described above on the same machine while guaranteeing isolation between containers. Our Docker Image (based on the official Dockerhub CentOS 7.3 and gcc 4.8) was created only once, and it included the `ctools` suite with its proper dependencies (Gammalib, Python, and C++ libraries) and the calibration IRFs files (see details on the requirements in Knödlseder et al. 2016).

3.2. AWS Elastic Cloud Computing

Amazon Elastic Cloud Computing 2 (EC2)[10] is a cloud service that offers computational power through objects called "instances." We can consider an instance as a computer node connected to the internet, with its own CPU, RAM, and disk space (called instance storage), as shown in Figure 1. These resources are fully maintained in the AWS Availability Zones, which are large server farms distributed across the world. One of the main advantages, in addition to the large available computational power, is that instances are fully maintained by AWS and the final user does not need to care about hardware configuration, maintenance, and general housekeeping. Moreover, the underlying hardware is regularly upgraded without any disruption of the service or user required actions.

Instances are acquired and fired up at need, depending on the number of realizations to be performed. When their work is completed instances are switched off, and all their content, including the instance storage, deleted. The AWS platform offers many different types of instances with various configurations in terms of CPU, RAM, and network performance. For our purpose, we chose spot instances (which are basically an unsold computation capacity supplied at very low prices) of type "m4x16large" that offer 64 Virtual CPUs (vCPUs) and 256 GB of RAM, which is thus sufficient to run at least 62 independent realizations part of a `ctools` simulation (considering 2 vCPU for the underlying operating system and Docker hypervisor). We decided to opt for the "m4x16large" EC2 instance type since it combines a reasonable ratio of vCPU and RAM while offering a best compromise between availability, when required as Spot, and price. Our EC2 cluster configuration here explained reaches a trade-off between the number of running instances, the cost per hour, and the number of Elastic Block Storage volumes (the hard disk equivalent) deployed in the region. We generally fired up a cluster of 30 of such instances so that we could run up to 1860 simultaneous realizations.

3.3. Amazon Simple Storage Service

AWS Simple Storage Service (S3)[11] is a service used to save large amounts of data with high consistency and durability, while affording a very fast access time. It resembles the commonly used file storage services, but it also offers advanced functionalities that allowed us to coordinate other cloud-based facilities (like a Lambda function, see Section 3.4). We used S3 to store data (input XML files and scripts required to run realizations, output FITS event files, and generally all results from our realizations) produced by the cluster of instances before they were switched off (see Section 3.2). We also used this service as a long-term storage by moving old data to a particular storage class called `Glacier`[12] that has lower maintenance costs and increased long-term durability of data. For this project we needed less than 1 GB to store all of the final results.

3.4. Amazon Lambda

`Lambda`[13] is a computer service that runs functions in response to some events triggered in the cloud (for example, a file upload to S3). The resources required to run functions and the triggers are automatically managed. We used AWS `Lambda` to coordinate the step between the upload of simulation inputs to S3 and the consequent EC2 cluster fire-up.

3.5. Amazon Simple Queue Service (SQS)

As we have seen a simulation is made of a thousand independent realizations, each with its own input parameters,

[10] https://aws.amazon.com/ec2/

[11] https://aws.amazon.com/S3/

[12] https://aws.amazon.com/glacier/

[13] https://aws.amazon.com/lambda/

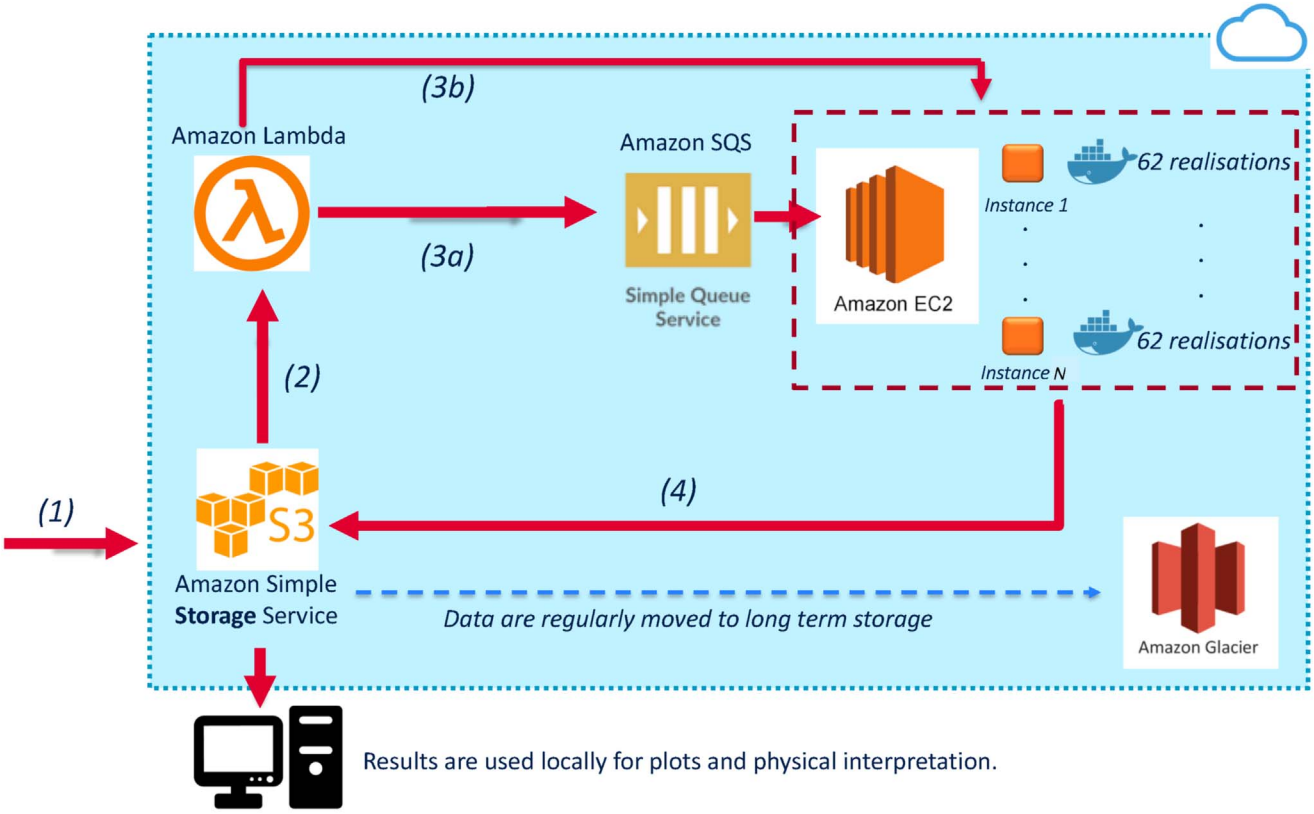

Figure 3. Cloud-based AWS architecture for parallel and distributed CTA simulations. This solution foresees the implementation of Amazon EC2 Service as computational power and Amazon S3 and Glacier as a baseline for the required storage. The Amazon Lambda and Amazon SQS services orchestrate the logical workflow of the system (see Section 3 for details).

so that a simulation could be thought as a table where each row contains the information to run a single realization (see Figure 2). In this scenario Amazon Simple Queue Service (SQS)[14] allows us to store this table and to distribute rows (technically called "messages") across the cluster in the common producer-consumer paradigm. In our approach the producer is the simulation table (stored only at the beginning of the whole simulation) with its large list of rows, while consumers are CPUs distributed across the EC2 cluster (see Figure 2). Each of them is in charge of completing one full realization. The possibility to exploit SQS provides a method to host queued messages and to reduce connectivity problem between consumers.

3.6. The AWS-based Cloud Architecture

As we described the concepts and services required to run a full simulation we are now able to detail the flow of data on the AWS architecture implemented for CTA (see Figure 3).

The distributed computation starts by (step 1) uploading a tarball file containing the XML file description of the source to simulate and the simulation table (as plain text file) in S3. As soon as the upload is completed a trigger is raised (step 2) and an Amazon Lambda function pushes the simulation table into the distributed and highly available first-in first-out queue SQS (step 3a). Then, it creates a homogeneous set of EC2 instances and fires them up, with the number of instances varying according to the overall size of the simulation (step 3b). This allows the distribution of realizations among many computers. When each EC2 instance is online, it automatically pulls up to 62 messages from the SQS queue. Each one of these messages contains the information necessary to run the realization on the node through execution of the Docker containers. Then, the EC2 instance waits for all containers to finish their computation and the outputs, temporarily saved on the local instance storage, are saved onto S3 (step 4). Processed data can then be recovered from S3 for a subsequent analysis and physical interpretation of the results.

[14] https://aws.amazon.com/sqs/

4. Results

In Table 2 we report the mean values and 1σ uncertainties of the simulation parameters obtained with $N = 1000$ independent realizations for the four input model fluxes. We consider a source detected with a high significance when TS $\geqslant 25$ (Mattox et al. 1996) and a low significance when $10 \leqslant$ TS < 25. The source will not be considered detected for TS < 10. We note that the mCrab source is not detected in all realizations for 5 hr and 10 hr exposure times. This is graphically shown in the distributions of TS values in Figure 4. Therefore, we shall only consider the case of 50 hr for the mCrab source, and the TS distribution cannot be modeled by a Gaussian. For each of the four input model fluxes, we show the distributions of the normalizations (Figure 5), spectral indices (Figure 6), TS values (Figure 7), and derived fluxes in the 0.1–100 TeV energy band (Figure 8).

As expected the spread in the parameter values depends strongly on the input flux. Indeed we can clearly see that the parameters are progressively more constrained as the input flux increases, so that the relative uncertainty of the output flux

Table 2
Mean Values and 1σ Uncertainties Based on $N = 1000$ Realizations

Model	Expo. (h)	k_0 (ph cm^{-2} s^{-1} MeV^{-1})	γ	TS	Events	Flux (erg cm^{-2} s^{-1})	$\Delta F/F$
Crab	5	$(5.70 \pm 0.03) \times 10^{-16}$	2.480 ± 0.005	$227\,288 \pm 1\,661$	$36\,833 \pm 201$	$(5.248 \pm 0.036) \times 10^{-10}$	0.0069
0.1Crab	5	$(5.70 \pm 0.11) \times 10^{-17}$	2.479 ± 0.016	$10\,213 \pm 291$	$3\,684 \pm 60$	$(5.26 \pm 0.12) \times 10^{-11}$	0.022
0.01Crab	5	$(5.74 \pm 0.60) \times 10^{-18}$	2.483 ± 0.072	301 ± 42	368 ± 20	$(5.26 \pm 0.44) \times 10^{-12}$	0.084
mCrab[a]	5	$(6.8 \pm 4.6) \times 10^{-19}$	2.6 ± 1.2	9 ± 7	37 ± 6	$(5.29 \pm 3.5) \times 10^{-13}$	0.67
mCrab[a]	10	$(6.3 \pm 3.3) \times 10^{-19}$	2.50 ± 0.65	14 ± 8	73 ± 9	$(5.34 \pm 2.2) \times 10^{-13}$	0.40
mCrab	50	$(5.9 \pm 1.4) \times 10^{-19}$	2.49 ± 0.13	68 ± 18	328 ± 18	$(5.30 \pm 0.83) \times 10^{-13}$	0.16

Note. Input values for the Crab spectrum are $k_0 = 5.7 \times 10^{-16}$ ph cm^{-2} s^{-1} MeV^{-1} and $\gamma = -2.48$; the expected 0.1–100 TeV band flux is 5.248×10^{-10} erg cm^{-2} s^{-1}. For the other sources a scaling factor of 0.1, 0.01, and 0.001 to k_0 is applied.
[a] The source is not detected in most of the realizations; see Figure 4.

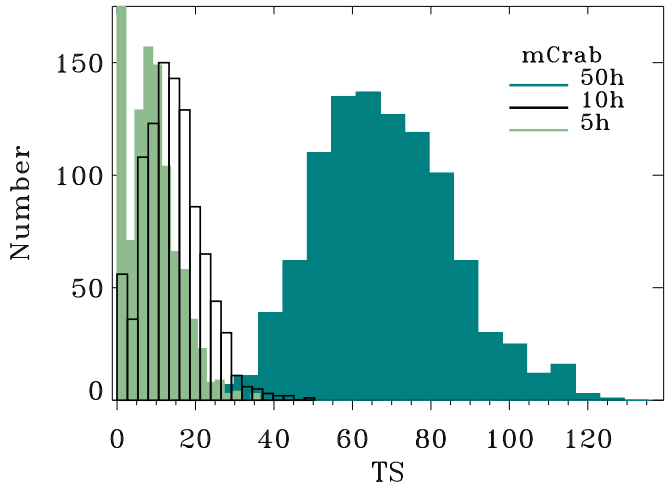

Figure 4. Comparison of the distributions of the test statistic TS values for the mCrab source as a function of the exposure time (5 hr, 10 hr, and 50 hr). The source is not detected in most realizations for 5 hr and 10 hr, and the TS distribution cannot be modeled by a Gaussian.

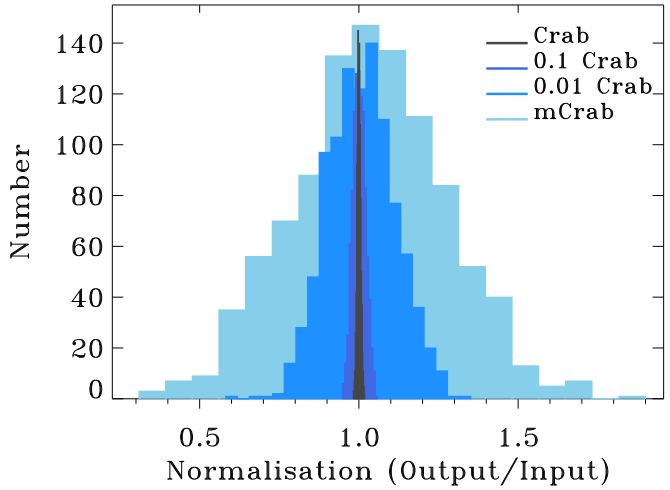

Figure 5. Distributions of the power-law fit normalization (column 3 of Table 2), as a function of the input flux. All distributions have been normalized to the input flux (see Section 2): 1 Crab (charcoal, in units of 5.7×10^{-16} ph cm^{-2} s^{-1} MeV^{-1}), 0.1 Crab (dodger blue, in units of 5.7×10^{-17} ph cm^{-2} s^{-1} MeV^{-1}), 0.01 Crab (navy blue, in units of 5.7×10^{-18}) ph cm^{-2} s^{-1} MeV^{-1}), and 1 mCrab (sky blue, in units of 5.7×10^{-19} ph cm^{-2} s^{-1} MeV^{-1}). Exposure times are 5 hr for all but the mCrab case, where 50 hr were used.

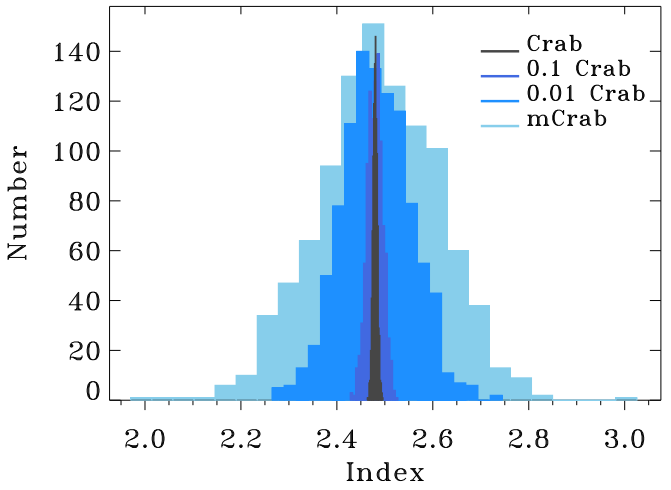

Figure 6. Distributions of the index in the power-law fit, as a function of the input flux (column 4 of Table 2).

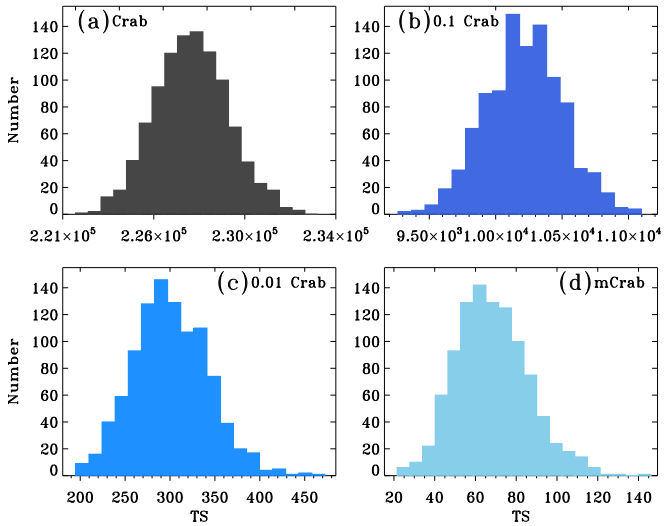

Figure 7. Distributions of the TS, as a function of the input flux (column 5 of Table 2); (a): 1 Crab, (b): 0.1 Crab, (c): 0.01 Crab, and (d): 1 mCrab.

decreases down to ∼16% for a mCrab and to within ∼0.7% for a Crab (see Table 2).

In other terms Figures 5–8 show how, due to the fact that individual realizations vary considerably in terms of derived parameter values and their statistical uncertainties (see in particular for the mCrab case, where the effect is more evident),

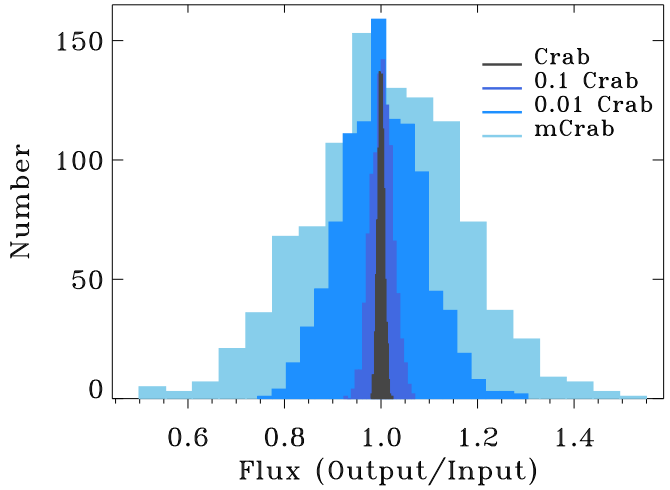

Figure 8. Distributions of the fluxes (column 7 of Table 2), normalized to the input flux: 1 Crab (in units of 5.248×10^{-10} erg cm^{-2} s^{-1}), 0.1 Crab (in units of 5.248×10^{-11} erg cm^{-2} s^{-1}), 0.01 Crab (in units of 5.248×10^{-12} erg cm^{-2} s^{-1}), and 1 mCrab (in units of 5.248×10^{-13} erg cm^{-2} s^{-1}).

a large number of realizations are always required in order to obtain average output parameters that are truly representative of the input spectrum. This is particularly critical for faint sources, since the background can become dominant. This, in turn, implies longer and longer run times for simulations of progressively fainter sources.

5. Budget Discussion

As reported in Table 1 the typical run time (single core) for a total of $N = 1000$ independent realizations varied from 8 hr (a Crab for 5 hr exposure time) to over 3 days (a mCrab, 50 hr exposure time). We further note that our test sources required a considerably simpler treatment than simulations of realistic astrophysical sources and lines of investigation, for which several more factors need to be taken into account.

The Galactic diffuse background is dominant for all sources close to the Galactic plane (e.g., ($|b| < 10°$). Based on our simulations performed on HESS J0632+057 (M. Chernyakova et al. 2019, in preparation), the run time can increase up to a factor of 5–10. Detailed simulations, including both event generation and a likelihood analysis, of a typical astrophysical Galactic source can then take months of CPU time.

The application of the energy dispersion matrix for sources requiring a detailed spectral treatment in the softer energies (e.g., $E \lesssim 500$ GeV) will be shown (Romano et al. 2018; Lamastra et al. 2019) to increase the run time by a factor of 5–10.

VHE astrophysicists are now also tackling the task of simulating a large populations of sources, e.g., active galactic nuclei, gamma-ray bursts (Ghirlanda et al. 2015), and binary sources. This task can quickly become prohibitive on less than a cluster of high-performance machines. Moreover in all cases the presence of contaminating sources in the 5° FOV considered for event extraction, a likely case in crowded regions, will scale the run time with the number of contaminating sources.

A further complicating issue is the storage of intermediate data and final data. For our four test sources the typical event file (the largest intermediate product of our pipeline) size was 22 MB for a 5 hr exposure, with the mCrab case reaching

Table 3
Budget of `ctools` Simulations for the Test Case (Section 2)

Costs (USD)	Local	AWS
Simulations	4	1.3
Storage	2.0[a]	5.0[b]
Maintenance	2.5	...
Total	8.5	6.3
Run times (hr)	172	0.5
Scaling factors	Local	AWS
Galactic background	×5–10	×5–10
Energy dispersion	×5–10	×5–10

Notes.
[a] Only considering temporary storage on HDs, which need to be backed up for the storage of final products and cleaned periodically from unused intermediate products.
[b] Including long-term storage and billing costs.

∼160 MB. The test case, as a whole, required a total storage space of 0.5 TB on a fast access disk. Realistic projects our group are currently pursuing are producing ∼300 MB event files and dozens of TB of intermediate products. For the average standalone machine this implies regular backing up, storing final products, and deleting unused intermediate products.

As a conclusion we report a comparison of the costs required for running our test case on a local machine and on AWS. Our local machine (Intel® Xeon® CPU E5-2620) costs about 7000 USD, considering both hard disk (HD) storage and uninterruptible power supply, which, assuming a mean time between failures of three years, implies an hourly cost of 0.022 USD hr^{-1} core^{-1}. The test case, therefore, cost about 4 USD, in terms of CPU, and took about 172 core hr to run. Table 3 shows (column 2, rows. 1–4) the costs for running the simulations locally, for storage, and for maintenance, respectively. These costs do not include the costs that the project needs to sustain for power and air conditioning or data management (creating queues for script running, cleaning up storage disks, and general housekeeping) during the simulations, generally in terms of human effort and permanent storage.

In comparison, on average, AWS spot instances in the cheapest availability zones cost about about 0.0078 USD hr^{-1} core^{-1}, so that the test case cost about 1.3 USD for the run and 5 USD for the storage (see Table 3, column 3, rows 1–4). This implies that use of AWS reduced the hourly cost per vCPU core by a factor of ∼3. The test case run was also performed successfully on AWS using our architecture based on a cluster of 60 m4.16xlarge spot instances in only ∼0.5 hr, which is a factor of 350 faster than locally. The combination of burst simulations (long but infrequent) and the possibility to exploit the unsold capacity on AWS cloud platform allows us to both reduce the required amount of time to run a full simulation while guaranteeing low cost being billed for just the used CPU time. In this way it is not necessary to pay in advance for a sitting idle local system. We note that the `ctools` suite could be run using the in-memory pipeline to avoid I/O for the temporary disk data storage of event files, further decreasing the overall cost. Clearly, for a small project, such as the test case, we presented a factor of 3 reduction in the CPU cost may not go a long way to justify the time spent to make the codes

parallel and distributable and to learn to use the cloud services; nor a factor of a few hundred in run time, if time is not of the essence. However, as Table 3 shows, other factors need to be taken into account when dealing with simulations of real astrophysical sources. While on AWS all intermediate processes are taken care by the Lambda functions when running simulations locally, the human effort required increases with the number of realizations being run. While it is indeed true that it is feasible to do many things with ctools without having a computer farm, the determination of the TS distribution to assess source detectability (for which a large number of realization is mandatory) require a considerable amount of computational power. This applies also to the simulation of large volumes of data or a complex analysis pipeline, such as needed for building the GPS catalog. In these end-to-end full simulations months of CPU time could be required, achievable with our proposed concept for a small amount of money and fully scalable in relation with the complexity of projects.

We are grateful to the referee for their review that improved the quality of our manuscript. We thank L. Foschini, J. Bregeon, G Maier, and K. Kosack for helpful discussions.

The authors acknowledge contribution from the grant INAF CTA–SKA, "Probing particle acceleration and γ-ray propagation with CTA and its precursors" (PI F. Tavecchio).

This research has made use of the CTA instrument response functions provided by the CTA Consortium and Observatory, see https://www.cta-observatory.org/science/cta-performance/ (version prod3b-v1) for more details.

This research made use of ctools, a community-developed analysis package for Imaging Air Cherenkov Telescope data. ctools is based on GammaLib, a community-developed toolbox for the high-level analysis of astronomical gamma-ray data.

We gratefully acknowledge financial support from the agencies and organizations listed here: http://www.cta-observatory.org/consortium_acknowledgments. This paper has gone through internal review by the CTA Consortium.

facilities: Cherenkov Telescope Array CTA.

software: ctools (Knödlseder et al. 2016).

ORCID iDs

M. Landoni https://orcid.org/0000-0003-2204-8112
P. Romano https://orcid.org/0000-0003-0258-7469
S. Vercellone https://orcid.org/0000-0003-1163-1396
F. Tavecchio https://orcid.org/0000-0003-0256-0995

References

Acharya, B. S., Actis, M., Aghajani, T., et al. 2013, APh, 43, 3
Actis, M., Agnetta, G., Aharonian, F., et al. 2011, ExA, 32, 193
Genoni, M., Landoni, M., Riva, M., et al. 2017, Proc. SPIE, 10329, 103290Z
Ghirlanda, G., Salvaterra, R., Ghisellini, G., et al. 2015, MNRAS, 448, 2514
Hinton, J. A., & Hofmann, W. 2009, ARA&A, 47, 523
Hofmann, W. 2017a, in AIP Conf. Ser. 1792, 6th International Symposium on High Energy Gamma-Ray Astronomy (Melville, NY: AIP), 020014
Hofmann, W. 2017b, Msngr, 168, 21
Knödlseder, J., Mayer, M., Deil, C., et al. 2016, A&A, 593, A1
Lamastra, A., Tavecchio, F., Romano, P., Landoni, M., & Vercellone, S. 2019, APh, submitted
Lemoine-Goumard, M. 2015, ICRC, 34, 12
Mattox, J. R., Bertsch, D. L., Chiang, J., et al. 1996, ApJ, 461, 396
Romano, P., Vercellone, S., Foschini, L., et al. 2018, MNRAS, 481, 5046
Williams, B. F., Olsen, K., Khan, R., Pirone, D., & Rosema, K. 2018, ApJS, 236, 4