Cloud-Based Urgent Computing for Forest Fire Spread Prediction under Data Uncertainties

Edigley Fraga, Ana Cortés, Tomàs Margalef, and Porfidio Hernández
Department of Computer Architecture and Operating Systems
Universitat Autònoma de Barcelona
Bellaterra, Spain
e-mail: edigley@gmail.com,{ana.cortes,tomas.margalef,porfidio.hernandez}@uab.cat

***Abstract*—Forest fires severely affect many ecosystems every year, leading to large environmental damages, casualties, and economic losses. Emerging and established technologies are used to help wildfire analysts determine fire behavior and spread, aiming at more accurate prediction results and efficient use of resources in fire fighting. We propose a novel forest fire spread prediction platform based on a proven two-stage prediction model devised to deal with input data uncertainties. The model is able to calibrate the unknown parameters based on the real observed data using an iterative process. Since this calibration is compute-intensive and due to the unpredictability of urgent computing needs, we exploit an elastic and scalable cloud-based solution platform implemented through coarse-grain parallel processing using a work queue.**

***Keywords*-cloud computing; data uncertainty; forest fires; urgent computing;**

I. Introduction

Fire is a natural element of many ecosystems, and even large wildfires are part of a defined disturbance regime [1]. For that reason, the challenge from both a prevention and a suppression point of view is to anticipate and reduce the spread potential of large wildfires and the succeeding risk to human lives, property, and land-use systems [2]. Wildfires have a relatively unpredictable nature as their spread can vary based on the flammable material and can differ by their extent and wind speeds. Forest fire prevention's strategies for detection and suppression have improved significantly through the years, both due to technological innovations and the adoption of various skills and methods.

Nowadays, wildfire researchers use technologies that integrate data on weather prediction, topography, and other factors to predict how fires spread. Forest fire prediction, prevention, and management measures have become increasingly important over the decades. Systems for wildfire prediction represent an essential asset to back up forest fire monitoring and extinction. They also serve to predict forest fire risks and to help in fire-control planning and resource allocation.

When dealing with the extinction phase, an accurate prediction of fire propagation is a critical issue to minimize its effects. Actually, while used in a fire extinction activity, a wildfire spread prediction is a *hard-deadline-driven task*. For instance, a complex wildfire simulation that could accurately predict the perimeter of a wildfire a couple of days ahead can drive firefighters to put firebreaks where they would be most effective to stop the fire propagation. In this particular case, an accurate prediction that comes up late compared to the actual event is useless to the task of fire suppression. These characteristics represent an urgent computing system, from which the simulation results are needed by relevant authorities in making timely informed decisions to mitigate financial losses, manage affected areas and reduce casualties [3]. In the mentioned forest fire propagation system, we can find the three urgent computing requirements:

a The computation operates under a strict deadline, after which the computation results may give little practical value ("*late results are useless*");
b The beginning of the event that demands the computation is unpredictable;
c The computation requires significant resource usage.

A solution must be *deadline-driven*, *on-demand provisioned*, and *scalable* to fulfill these requirements. To deal with them, High-Performance Computing (HPC) community used to rely on dedicated high-end clusters, supercomputers, or distributed computing platforms [4].

As an enabling technology, cloud computing allows new strategies to deal with the urgent computing challenge. Considering cloud computing characteristics of on-demand provisioning, immediate scalability, and abundant offer of resources (b), one might consider it a natural choice as a platform to run urgent computing applications. Regarding strict deadline (a) and resource usage (c) requirements, although in the early days cloud offerings suffered to run HPC applications, today they are an alternative to on-premise clusters [5].

An important aspect of any system is the incurred cost. Traditional HPC applications involve a high Total Cost of Ownership (TCO) on the subjacent platform, but cloud-based solutions allow access to supercomputing-like features at the cost of a few dollars per hour. This is particularly important for fire propagation systems due to 1) the seasonality of the wildfire occurrences and 2) because forest fire prevention services or fire brigades usually can't afford the TCO to keep an infrastructure idle until eventually needed by an urgent computation.

The use of simulators for forest fire propagation requires

This research has been supported by MINECO-Spain under contract TIN2017-84553-C2-1-R, MICIN-Spain under contract PID2020-113614RB-C21 and by the Catalan government under grant 2017-SGR-313.

sufficient time for the processing, precise fuel model data, and accurate knowledge of small and large-scale interaction of weather and topography [2]. To start a simulation, a plethora of input parameters is necessary. They include spatial data describing the elevation, slope, aspect, and fuel type. Fig 1 illustrates the input data commonly used for fire simulations.

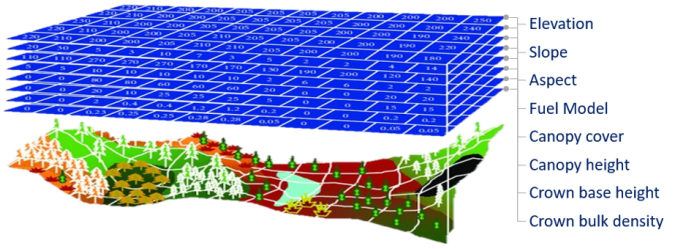

Fig. 1. Fuel and topographic grid data used for wildfire simulations.

In reality, the input data describing the actual scenario where the fire is taking place is usually subject to high levels of uncertainty that represent a serious drawback for the correctness of the prediction [6]. Any designed solution faces a real challenge while accurately predicting a fire spread in a scenario with uncertainties regarding input data, under a strict deadline constraint, and in a cost-effective way.

Taking into account new advancements in cloud computing offerings, it is possible to achieve more accurate prediction results in less time even for larger fire occurrences when compared to traditional HPC-based solutions. But besides guaranteeing short execution time (performance) one must also take into consideration the total simulation cost (total expense) involved in the designed solution [7]. As a consequence, we propose a performance-efficient and cost-effective cloud resource management platform built upon a proven methodology for forest fire prediction.

So far, on-premises HPC solutions were usually built under the restriction of the scarcity of available computing resources [8]–[10] or on multi-million dollar supercomputing infrastructures [11]. In contrast, one of cloud computing characteristics is abundance, which results in the illusion of infinite computing resources available on demand [12]. Therefore, our working hypothesis is that a solution based on cloud offerings allows improving the accuracy of fire propagation prediction results. Obviously, some challenges arise since there are still some gaps between HPC and cloud paradigms regarding computing power and communication efficiency [13].

The remainder of this document is organized as follows. Related work is discussed in Section II and a brief introduction of the FARSITE simulator is presented in section II-A. Section II-B describes the two-stage prediction method used to deal with input-data uncertainty. Section III presents the proposed cloud-based solution. A proof-of-concept implementation is discussed in Section IV. Finally, some concluding remarks are given in Section V.

II. Related Works

Abdalhaq [14] proposed a two-stage methodology to calibrate the input parameters in an adjustment phase so that the calibrated parameters are used in the prediction stage to improve the quality of the predictions. Cencerrado [9] applied Genetic Algorithm as the calibration technique in the adjustment phase, which requires the execution of many simulations to generate the best-calibrated set of input parameters. Similar work was also carried over by Mendez-Garabetti et al. [15]. Cencerrado also devised one strategy based on *Decision Trees* to identify long-running execution individuals of a fire spread simulation. Such a strategy was the base for a classification method that allows estimating in advance the execution time of a simulation given a certain set of input parameters.

More recently, Artés [8] proposed and evaluated a set of resource allocation policies to assign more computing resources to estimated long-running executions and fewer resources to the fast ones, allowing to reduce the adjustment stage time to a more acceptable deadline. That was possible due to the use of a parallel version of the FARSITE model that could reduce long-running execution times by 35%. To work in a time-constrained fashion, a hybrid MPI-OpenMP application based on the Master-Worker paradigm was developed to take advantage of the execution in a parallel HPC cluster environment. Fraga [16] proposed, implemented, and evaluated an early adaptive evaluation strategy to further speed up the calibration process, reducing the overall calibration time by 60%. The two-stage framework has been proved to be a good methodology to deal with the input data uncertainties, and it is leveraged in this current work.

Altintas et al. [11] conducted the comprehensive WIFIRE Firemap project, which is a dynamic data-driven system to predict wildfire progress through data analysis that is relayed through map visualizations. FARSITE is one of the leveraged fire spread simulator models coupled with a wind simulator. Compute-intensive tasks are executed in parallel on distributed computing environments and public cloud offerings.

Kalabokidis and others [17] implemented a cloud application composed of wildfire risk and spread simulation service. End users access the application in a software as a service delivery model, being charged for their consumed processing time during the actual wildfire simulation period. The application presents the flexibility to scale up or down the number of computing nodes needed for the requested processing depending on the number of simultaneous users. Although being a step toward the spread of adoption of simulation techniques to local fire agencies, the solution does not address the issues related to input data uncertainties, neither was developed to be applied in urgent computing firefighting scenarios.

A. Forest Fire Spread Simulator

In the field of forest fire behavior modeling, there are fire propagation simulators based on different physical models, whose main objective is to predict the fire evolution. Among those, FARSITE [18] is a well-known fire growth simulation modeling system that uses spatial information on topography

and fuels along with weather and wind inputs. It makes it possible to compute wildfire growth and behavior for long periods under heterogeneous conditions of terrain, fuels, and weather. It also incorporates existing models for surface fire, crown fire, spotting, post-frontal combustion, and fire acceleration into a two-dimensional fire growth model. Its simulation generates a sequence of fire perimeters representing the growth of a fire under a given input condition. For that purpose, it incorporates, among others, the simple but effective Rothermel's surface fire spread behavior model [19] along with Huygens's Principle of wave propagation [18]. A wildfire simulation is a process inherently complex, from which a long execution time for an individual simulation is not atypical.

B. Two-Stage Prediction Method

Usually, to predict forest fire behavior, a simulator takes the initial state of the fire front perimeter (P_0) along with other parameters as input. As output, the simulator then returns the fire front spread prediction for a later instant in time ($\hat{P}_1$). After comparing the simulation result with the real advanced fire front (P_1), the predicted fire line tends to differ from the actual one. Besides the natural phenomena modeling complexity uncertainty, the reason for this mismatch is that the *Prediction Stage* in the classic scheme calculation is based solely on a single set of input parameters, affected by the aforementioned data uncertainty. To overcome this drawback, a simulator-independent data-driven prediction scheme was proposed to calibrate model input parameters [9]. Fig. 2 illustrates this Two-Stage Prediction Method.

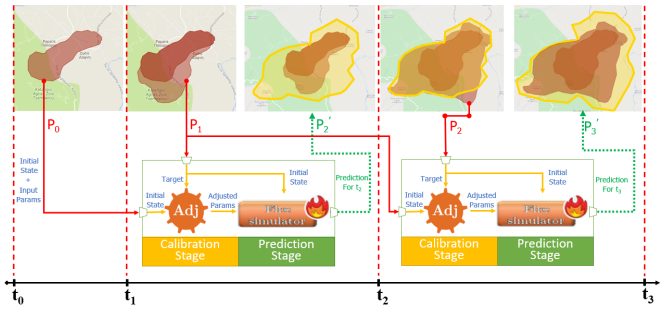

Fig. 2. Two-Stage Prediction Method.

Introducing a *Calibration Stage*, the set of input parameters is adjusted before every prediction step. Thus, the solution comes from reversing the problem, coming up with a parameter configuration such that the fire simulator would produce predictions that match the actual fire behavior. After detecting the simulator input that better reproduces the observed fire propagation, the same set of parameters is used to describe the conditions for the next prediction ($\hat{P}_2$), assuming that meteorological circumstances remain constant during the next prediction interval. Prediction is then the result of a series of adjusted input configurations. The process can be applied again for subsequent fire perimeters ($\hat{P}_3$), (P_3) and so on.

As a data-driven prediction scheme, to enhance the quality of the predictions, the two-stage method is applied continuously. Providing calibrated parameters at different time intervals and taking advantage of observed fire behavior, this approach has been proven to be appropriate to enhance the quality of the predictions. In particular, a Genetic Algorithm (GA) based adjustment technique gives accurate results.

III. A Cloud-Based Urgent Computing Solution

Fig. 3 shows a high-level architecture for a **C**loud-based **u**rgent **Co**mputing (dubbed *CuCo*) solution. Apart from the *Prediction Engine* based on the two-stage prediction method, there is also an *Optimized Cost Estimator*, capable of preparing a resource allocation plan that minimizes cost while maintaining the prediction deadline.

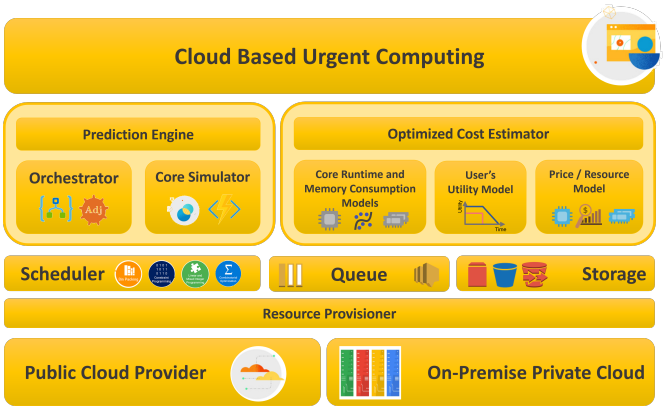

Fig. 3. High-Level Architecture of CuCo's Solution.

The main components are the following:

- **Prediction Engine**: it is based on the two-stage prediction method, running a calibration phase before the actual fire spread prediction. Currently, the calibration's *Orchestrator* is based on an iterative data-driven compute-intensive Genetic Algorithm that runs FARSITE as the *Core Simulator*.
- **Optimized Cost Estimator**: a domain-specific module, tailored to statistically model the core simulator, specify the available computational resources properties and determine the user preferences through strict deadlines or utility functions.
  - **Core Runtime and Memory Consumption Models**: to estimate the execution time of a fire spread simulation, it is necessary to perform a large set of executions of the underlying simulator on the target architecture and then analyze its behavior from the obtained results. We use the results presented in [8] and [9] as characterization models.
  - **User's Utility Model**: to capture user's satisfaction over prediction time and monetary cost of the overall fire spread prediction, we propose to foster utility functions, as they are commonly used to communicate the value of work and other quality of service aspects such as its timely completion [20].
  - **Price / Resource Model**: in this work, we consider an *Infrastructure-as-a-Service* (IaaS) cloud system, in

which a number of data centers deliver on-demand storage and compute capacities over the Internet. These computational resources are provided in the form of an abstract unit of compute and storage called *Virtual Machine* (VM for short), object storage, or remote file systems volumes. In a cloud, VMs are offered in different types, each of which has different characteristics such as different numbers of CPUs, amount of memory, and network bandwidths capacity.

- **Storage**: as an urgent computing solution, it is necessary to have on-demand access to the input data. It is needed a persistent repository consisting of all the static data for the areas with a greater probability of forest fire occurrence (those with high fire danger indices). For these static data, as a data lake, the proposed solution leverage object storage service that offers industry-leading scalability, data availability, security, and performance.
- **Queue**: it integrates the different components providing an Application Programming Interface (API) to create queues as well as send, receive, and delete messages. Messages are used to represent simulation tasks; queue states to trigger orchestration or scheduling activities.
- **Scheduler**: its function is to assign resources to perform computing tasks. It determines which resources are valid placements for each task according to their constraints.
- **Resource Provisioner**: it is responsible for providing the virtual resources in which the actual simulation will be run. After understanding the memory consumption and runtime of the core simulator, a suitable memory or compute-intensive pool of instances can be provisioned in order to overcome this potential point of congestion.

IV. Proof-of-Concept Implementation

As a proof-of-concept implementation, we decided to leverage well-established cloud tools, favoring a decoupled solution that can be easily deployed on a public cloud or on-premise infrastructure. Considering Fig. 3, the orchestrating component is responsible for generating all the compute demands based on a stream of GA generations. The *Core Simulator* is the FARSITE simulation model. The *Scheduler* function is delegated to the Kubernetes orchestration engine whereas the *Resource Provisioner* is linked to the AWS EC2 infrastructure. Docker is the container runtime engine of choice.

FARSITE simulator has been isolated as a cloud component, deployed initially as a containerized application. The way to deal with the data acquisition latency problem is to rely on a well-established scalable object-storage service to allow the efficient parallel retrieval and storage of input and output data.

A. Kubernetes

A working Kubernetes deployment is called a cluster, including the control plane and the compute machines, or nodes.

Control plane: it manages the worker *nodes* and the *pods* in the cluster. It includes API, scheduler, and controller modules, apart from *etcd*, a consistent and highly-available key-value store used as Kubernetes' backing store for all cluster data.

Pods: A pod is the smallest/simplest unit in the Kubernetes object model, representing a single instance of an application. It can be a container or a series of tightly coupled containers, along with options that govern how the containers are run.

Nodes: A Kubernetes *cluster* needs at least one compute node, but will normally have many. *Pods* are scheduled and orchestrated to run on *nodes*. A direct way to scale up the capacity of a cluster is to add more *nodes*.

Container runtime: to run the containers, each compute *node* has a container runtime engine. Kubernetes supports *Open Container Initiative*-compliant runtimes. *Docker* has been chosen due to its consistent and standard binary format.

Container registry: the container images that Kubernetes relies on are stored in a container registry. This can be a locally-configured registry or a third-party one.

B. Underlying Virtual Infrastructure

Kubernetes can be run on bare metal servers, virtual machines, public cloud providers, private clouds, and hybrid cloud environments. One of Kubernetes's key advantages is that it works on many different kinds of underlying infrastructure, being a perfect fit for CuCo's architecture.

Amazon Elastic Compute Cloud (EC2) is the part of Amazon's cloud-computing platform that allows users to rent VMs (called "instances"). A user can create, launch, and terminate instances as needed, paying by the second for active servers. Like other providers, Amazon offers a plethora of pricing and service model. **Spot Instances** are spare compute capacity in the AWS cloud available at up to 90% discount compared to **On-Demand** prices. As a trade-off, AWS offers no SLA on these instances and customers take the risk that it can be interrupted at any moment. Speed-ups obtained through works from Artés [8] and Fraga [16] made it possible to take the mentioned risk. Considering this, we have chosen the Spot option as the most appropriate one due to its cost-effectiveness.

C. CuCo's Implementation Model

Supporting the driving GA *Prediction Engine Orchestrator*, we use a simple Kubernetes Job that performs a coarse parallel processing using a work queue.

Integration using a message queue service: the initial population is randomly generated and for each one of its individuals a task definition is created and pushed to the task queue. We opted for using *Simple Queue Service* from Amazon AWS, although any queue service could be used.

Kubernetes Job: the job starts several pods, and each of them takes one task from the message queue, processes it, and repeats until the end of the queue is reached. Once the task queue is empty, a new generation can be computed and the message queue can be filled up again. The process is repeated until the pre-defined number of generations is reached. In the end, the best individuals are made available in the output bucket and they can then be used for the *Prediction Stage*.

Filling the queue with tasks: one task is a scenario describing parameters for every single FARSITE's execution, representing a GA individual. A message must include the

TABLE I
AWS EC2 COMPUTE INSTANCES FAMILY

Model	Number of vCPU	Memory (GiB)	Cost per Hour
c5.large	2	4	$ 0.085
c5.xlarge	4	8	$ 0.17
c5.2xlarge	8	16	$ 0.34
c5.4large	16	32	$ 0.68
c5.9large	36	72	$ 1.53
c5.18xlarge	72	144	$ 3.06

input bucket and the input directory name. Likewise, the output bucket needs to be informed, accompanied by the name of the directory where output files will be pushed to.

Dockerizing FARSITE: A *Docker* template has been prepared to build a containerized version of FARSITE. In its first execution, the container attempts to consume a message from the task queue and, in case there is at least one task, proceeds to run the scenario and at the end push the results to the output directory defined in the received message for the scenario being executed. At the end of the execution, the message is permanently deleted from the queue.

Reconciling: Once the queue is empty, the Genetic Algorithm can take back the control of the execution and, if the evolution is not complete yet, applies the genetic operators resulting in a new generation. The queue is filled up again and Kubernetes takes the control in order to deploy, start, and at the end terminate all the pods used to run the individuals.

Fig. 4 illustrates the interaction flow between the main CuCo's components for the *Calibration Stage*, the most compute and time-consuming one. Each interaction is accompanied by a number representing its order in the flow. First of all, in CuCo's UI, the user (*wildfire analyst*) informs the scenario's configuration **(1)**, with all the static data mentioned in Fig. 1 being available in a storage service (*Amazon Simple Storage Service* - S3, in this current implementation). For each generation, the *Optimized Cost Estimator* is consulted **(2)** and a resource allocation plan is defined **(3)**. The list of resources to be acquired is then informed to **(4)** and provisioned by the *Resource Provisioner* that, in its turn, request **(5)**, receives **(6)**, and passes them to the *Genetic Algorithm Orchestrator* **(7)**. The orchestrator can finally provide the resources to the *Scheduler* **(8)** that starts the *Kubernetes Job* **(9)**. The queue is then filled up with the tasks **(10)** used to define each FARSITE's scenario that will be consumed **(11)** by the *pods* running in the *Kubernetes cluster*. Calibration results are pushed to the output bucket **(12)** until the end of the *Calibration Stage*. Once the calibration finishes, the adjusted params are passed on to the *Prediction Stage* **(13)** that runs the actual prediction on a special high-performance container **(14)**. In the end, the prediction result **(15)** is made available to the *wildfire analyst*.

D. Cost Estimator: The Cost-Performance Trade-Off

We consider the FARSITE characterization performed in [8] and [9] to define the most cost-effective AWS EC2 instance types to be used as the underlying virtual infrastructure. As a result, the *Compute Optimized* family listed in Table I has been

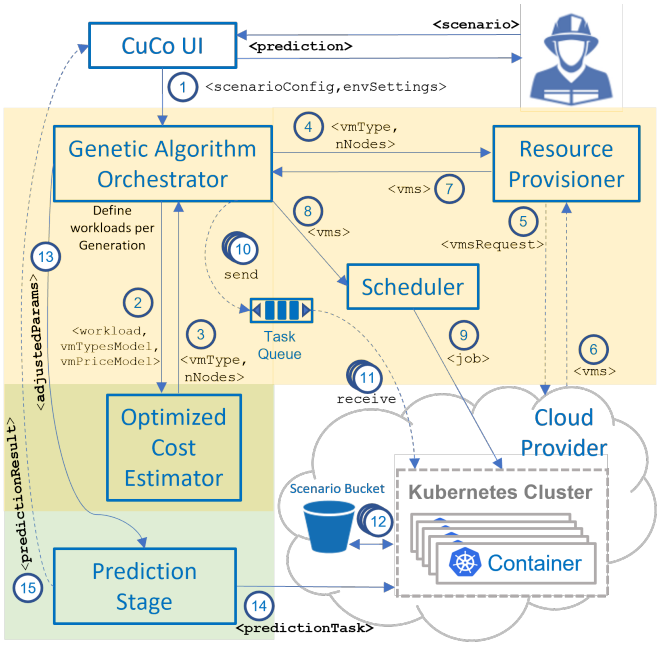

Fig. 4. Cuco's components interaction flow

chosen. The *Optimized Cost Estimator* is fed with these prices and resource characteristics and, based on the *Core Runtime and Memory Consumption Models* (see Fig. 3), generates as output the *average cost per individual* per *calibration time*. In Fig. 5 we can see the result for configuration spanning 1, 2, 4, 8, 16, and 32 nodes in the Kubernetes cluster.

The final step is to decide which configuration will be used based on the user's preference. We model the user's preference as a utility function, a mathematical function that ranks alternatives according to their utility to an individual. A commonly used utility function is the simplified version of the Cobb-Douglas production function $u(x_1, x_2) = x_1^{b_1} x_2^{b_2}$, where b_1 and b_2 are positive numbers describing the preferences of the consumer. Applying it for $b_1 = \frac{1}{2}$ and $b_2 = \frac{1}{2}$, i.e. the user is equally concerned with price and runtime, the resulting ranking of preference is shown in Table II. There are $6 \times 6 = 36$ configurations in total (*the number of instance types* $\times$ *the number of options for cluster size*). Considering the results, the configuration that gives more utility to user preferences is the one with 4 nodes of type c5.2xlarge. For different parameters of the utility function, a new ranking is provided, but it is up to the user, i.e. the *wildfire analyst*, to inform his/her preferences. The *Resource Provisioner* module can provision the corresponding virtual resources and then trigger Kubernetes to perform the scheduling task.

V. CONCLUSION

In this work we propose a novel forest fire spread prediction cloud platform based on a proven two-stage prediction method devised to deal with input data uncertainties. Based on the performance and cost estimation, together with the user preferences modeled as a utility function, the platform is able

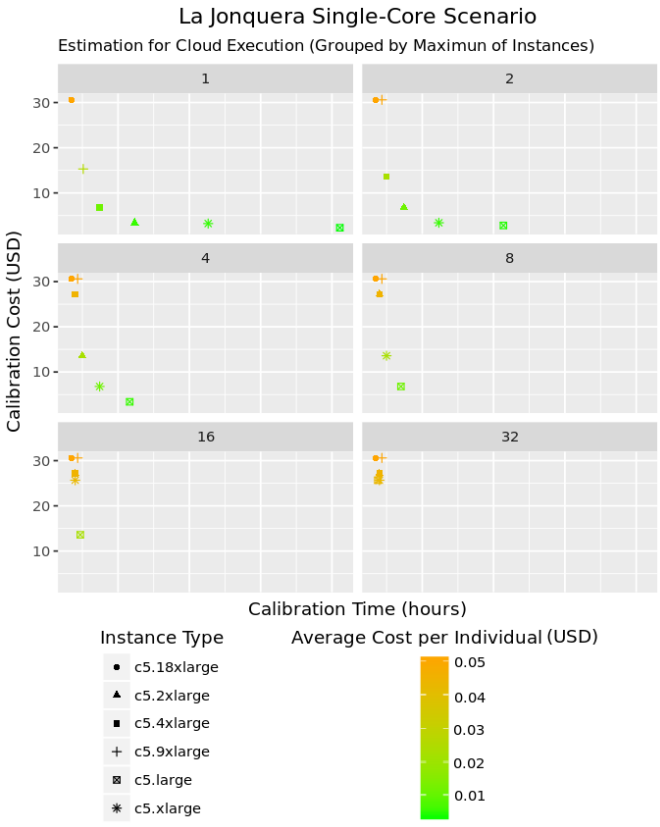

Fig. 5. La Jonquera cloud execution cost for the calibration phase.

TABLE II
RESULTING RANKING OF PREFERENCE TO RUN THE CALIBRATION

Ranking	Utility	Number of Nodes	Model
1	0.5529	4	c5.2xlarge
2	0.5037	2	c5.xlarge
3	0.4976	1	c5.xlarge
4	0.4593	16	c5.large
5	0.3828	2	c5.2xlarge
...	...	...	...
35	0.0847	2	c5.large
36	0.0603	1	c5.large

to suggest, allocate, and run a cloud virtual infrastructure that better satisfy the user's preferences and the workload demands.

Even for HPC applications, when dealing with cloud computing flexibility, we can explore new models and possibilities enabled by the economies of scale. A cluster-optimized application might not be always suitable for a cloud environment, then it is crucial to make cloud-friendly design choices.

Nevertheless, considering that HPC physical machines offer more performance than compute-optimized cloud instances, hybrid approaches are also worthy of further investigation. One point of future improvement is to consider different user objectives as, for example, accelerating application runtime within a given budget constraint instead of simply complying with a time constraint. We also plan to perform a comprehensive scalability analysis of CuCo's platform considering different workloads for persisting wildfires.

REFERENCES

[1] J. San-Miguel-Ayanz, J. M. Moreno, and A. Camia, "Analysis of large fires in european mediterranean landscapes: Lessons learned and perspectives," *Forest Ecology and Management*, vol. 294, pp. 11 – 22, 2013, the Mega-fire reality.

[2] P. Costa, M. Castellnou, A. Larrañaga, M. Miralles, and D. Kraus, *Prevention of Large Wildfires using the Fire Types Concept*. European Forest Institute, Jan 2011.

[3] S. H. Leong and D. Kranzlmüller, "Towards a general definition of urgent computing," *Procedia Computer Science*, vol. 51, pp. 2337 – 2346, 2015, proc. of the International Conference on Computational Science, ICCS 2015.

[4] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[5] A. Gupta and D. Milojicic, "Evaluation of HPC applications on cloud," in *2011 Sixth Open Cirrus Summit*, Oct 2011, pp. 22–26.

[6] M. P. Thompson and D. E. Calkin, "Uncertainty and risk in wildland fire management: A review," *Journal of Environmental Management*, vol. 92, no. 8, pp. 1895 – 1909, 2011.

[7] A. G. Carlyle, S. L. Harrell, and P. M. Smith, "Cost-effective HPC: The community or the cloud?" in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, Nov 2010, pp. 169–176.

[8] T. Artés, A. Cencerrado, A. Cortes, and T. Margalef, "Relieving the effects of uncertainty in forest fire spread prediction by hybrid mpi-openmp parallel strategies," *proc. of the International Conference on Computational Science, ICCS 2013*, vol. 13, no. 2, pp. 2277–2287, 2013.

[9] A. Cencerrado, A. Cortés, and T. Margalef, "Genetic algorithm characterization for the quality assessment of forest fire spread prediction," *Procedia Computer Science*, vol. 9, pp. 312 – 320, 2012, proc. of the International Conference on Computational Science, ICCS 2012.

[10] G. Bianchini, P. Caymes-Scutari, and M. Méndez-Garabetti, "Evolutionary-statistical system: A parallel method for improving forest fire spread prediction," *Journal of Computational Science*, vol. 6, pp. 58 – 66, 2015.

[11] I. Altintas, J. Block, R. de Callafon, D. Crawl, C. Cowart, A. Gupta, M. Nguyen, H.-W. Braun, J. Schulze, M. Gollner, A. Trouve, and L. Smarr, "Towards an integrated cyberinfrastructure for scalable data-driven monitoring, dynamic prediction and resilience of wildfires," *Procedia Computer Science*, vol. 51, pp. 1633 – 1642, 2015, proc. of the International Conference On Computational Science, ICCS 2015.

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, Apr 2010.

[13] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Computing Surveys*, vol. 51, 10 2017.

[14] B. Abdalhaq, A. Cortés, T. Margalef, and E. Luque, "Enhancing wildland fire prediction on cluster systems applying evolutionary optimization techniques," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 61 – 67, 2005.

[15] M. Méndez-Garabetti, G. Bianchini, M. L. Tardivo, and P. Caymes-Scutari, "Comparative analysis of performance and quality of prediction between ESS and ESS-IM," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 314, no. C, p. 45–60, Jun. 2015.

[16] E. Fraga, A. Cortés, A. Cencerrado, P. Hernández, and T. Margalef, "Early adaptive evaluation scheme for data-driven calibration in forest fire spread prediction," in *Computational Science – ICCS 2020*. Cham: Springer International Publishing, 2020, pp. 17–30.

[17] K. Kalabokidis, N. Athanasis, C. Vasilakos, and P. Palaiologou, "Porting of a wildfire risk and fire spread application into a cloud computing environment," *International Journal of Geographical Information Science*, vol. 28, no. 3, pp. 541–552, 2014.

[18] M. Finney, "FARSITE: Fire Area Simulator-Model. Development and Evaluation," *USDA Forest Service Research Paper, RMRS-RP-4*, 1998.

[19] R. Rothermel, *A mathematical model for predicting fire spread in wildland fuels*, ser. USDA Forest Service research paper INT. Intermountain Forest & Range Experiment Station, U.S. Dept. of Agriculture, 1972.

[20] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility functions in autonomic systems," in *International Conference on Autonomic Computing, 2004. Proceedings.*, May 2004, pp. 70–77.