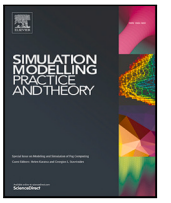
Simulation Modelling Practice and Theory 134 (2024) 102931

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat



A methodology for performance estimation of bot-based applications for natural disasters

Luis Veas-Castillo [a], Juan Ovando-Leon [c], Carolina Bonacic [c], Veronica Gil-Costa [b,*], Mauricio Marin [c]

[a] Universidad Austral de Chile, Valdivia, Chile
[b] Universidad Nacional de San Luis, San Luis, Argentina
[c] Universidad de Santiago de Chile, Santiago, Chile

ARTICLE INFO

Keywords:
Bot-based applications
Performance estimation
Model and simulation-based methodology

ABSTRACT

Natural disasters drastically impact the society, causing emotional disorders as well as serious accidents that can lead to death. These kinds of disasters cause serious damage in computer and communications systems, due to the complete or partial destruction of the infrastructure, causing software applications that actually run on those infrastructures to crash. Additionally, these software applications have to provide a stable service to a large number of users and support unpredictable peaks of workloads. In this work, we propose a methodology to predict the performance of software applications designed for emergency situations when a natural disaster strikes. The applications are deployed on a distributed platform formed of commodity hardware usually available from universities, using container technology and container orchestration. We also present a specification language to formalize the definition and interaction between the components, services and the computing resources used to deploy the applications. Our proposal allows to predict computing performance based on the modeling and simulation of the different components deployed on a distributed computing platform combined with machine learning techniques. We evaluate our proposal under different scenarios, and we compare the results obtained by our proposal and by actual implementations of two applications deployed in a distributed computing infrastructure. Results show that our proposal can predict the performance of the applications with an error between 2% and 7%.

1. Introduction

After a natural disaster strikes, people and different institutions collaborate to provide help as soon as possible to the victims. Emergency response teams composed of spontaneous volunteer and experts have a key role in helping to reduce the impact of the natural disaster. These teams use applications specially designed to coordinate the tasks and the volunteers. The applications must be deployed on a distributed computing platform formed of commodity hardware usually available from universities[1] [1–3]. Moreover, these applications are typically based on social computing concepts like instant text messaging, sharing photos and humanitarian concepts such as volunteering. These applications form an ecosystem that allow information to be collected in real time or can help improving the management of the emergency and assistance to victims. Additionally, these applications typically run on complex

* Corresponding author.
E-mail addresses: luis.veasc@inf.uach.cl (L. Veas-Castillo), juan.ovando@usach.cl (J. Ovando-Leon), carolina.bonacic@usach.cl (C. Bonacic), gvcosta@unsl.edu.ar (V. Gil-Costa), mauricio.marin@usach.cl (M. Marin).

[1] https://www.reuna.cl/infraestructura-digital/#red-nacional

https://doi.org/10.1016/j.simpat.2024.102931

Available online 5 April 2024

computational infrastructures that range from clusters of processors to smartphones, with communication supported by different networks and protocols. The scalability to thousands or even millions of users is a relevant issue to be considered when designing these applications as they are not expected to collapse when they are mostly needed.

In this work, we present an methodology aimed at predicting the performance of bot-based applications specifically designed for emergency situations. In particular, for emergency situations caused by natural disasters. Our proposal combines modeling and simulation with machine learning techniques to estimate the performance of the applications. Furthermore, our methodology incorporates a specification language used to formalize these models, ensuring precision and clarity in the analysis. By integrating these techniques and tools, we aim to provide a robust methodology for accurately predicting the behavior and performance of emergency response software applications in crisis scenarios.

1.1. Research challenges

The efficient design, implementation, and deployment of bot-based applications devised to be used in natural disaster scenarios present significant challenges, including resource allocation, infrastructure management, and the need for extensive expertise in distributed and scalable systems architecture. Several factors contribute to the increased complexity of developing efficient applications and computing systems for such environments:

- Infrastructure Failures: natural disasters often lead to communication and processor infrastructure failures, impacting the reliability and availability of applications [4].
- Abrupt increase in communications and social media applications usage: disasters result in a sudden increase in communications and usage of social media applications. Therefore, it is important to have a robust deployment of the applications to handle the increase of traffic [5].
- Mobility of users: users accessing social media and support bot-based applications during disasters are often mobile, requiring applications to be adaptive and accessible across different devices [6,7].
- Design for stressful environments: applications must be designed for users operating under stress during emergency situations [8–10].
- Coexistence of multiple applications: different applications must coexist on distributed platforms connected by high-speed networks, requiring efficient resource allocation and management strategies [11,12].

Given these challenges, the design and implementation of such software applications require significant time and resources. In this context, our proposed methodology is a valuable tool for architects and software developers responsible for designing, implementing, and deploying new applications or services within existing applications. We use as case study two applications that can be accesses through mobile devices and are designed to be used during stressful situations. Our proposal allows to evaluate various scenarios varying workloads, with and without failures, considering different applications running at the same time and different user request rates, to facilitate the design of algorithms and data structures without directly modifying real applications.

1.2. Contribution

In this work, we introduce a comprehensive methodology for developing discrete simulation models aimed at predicting the performance of applications designed for post-natural disaster usage. Our proposed methodology allows to address the challenges described above by modeling and simulating the services and components of different applications executed at the same time and deployed on a distributed platform comprising commodity hardware available from public institutions such as universities. The methodology has five steps. In the first step we define the attributes and criteria that applications must satisfy. In the second step, we run benchmarks to characterizes the behavior of the applications. Then, we model the applications and the platform used to deploy them. In the fourth step, we define the task flow of the applications. Finally, we simulate the applications deployed on the computing platform. To enhance the accuracy of our predictions, we integrate simulation with various machine learning techniques, including Simple Linear Regression (SLR), Multiple Linear Regression (MLR), Neural Networks, and Support Vector Machine (SVM). This combined approach allows us to analyze and estimate performance metrics effectively. Additionally, as we mention before, we introduce a specification language that formalizes the definition and interaction between the components within the applications. This language helps to ensure clarity and precision in the modeling process, facilitating better understanding and management of the applications performance under different situations.

We evaluate our proposed methodology with two bot-based applications, namely Ayni and Rimay, which serve as our case studies [13]. Ayni is responsible for managing and assigning tasks to volunteers, while Rimay registers volunteers, manages campaigns, and handles emergency tasks. These applications were specifically designed for the Chilean population in collaboration with national organizations. Results show that our proposed methodology is capable of estimating the performance of the application with a maximum error of 7.7%.

1.3. Outline

The remainder of this paper is organized as follows. In Section 2 we present related work. In Section 3 we present our computing platform and the applications used as case study for our proposal methodology. Section 4 presents the proposed methodology. Section 5 introduces a specification language designed for modeling distributed computation platforms utilized in deploying applications tailored for post-natural disaster scenarios. Section 6 presents the experimental results. Section 7 discuss the challenges and the limitations of our proposal and Section 8 concludes the paper.

2. Related work

This section summarizes a series of works related to large-scale infrastructure performance prediction based on machine learning and computer simulation.

The work presented in [14] studies the prediction of the execution time and the computational resources consumed on biomedical applications, using decision trees [15] and regression models [16]. Works such as [17,18] compare analytical models and machine learning models (*Linear Regression*, *Support Vector Machine* and *Random Forest*) for performance prediction of applications that use GPU. They conclude that analytical models give good results but require too much knowledge of the system. On the other hand, machine learning models, despite not obtaining such good results, do not require complete knowledge of the system, which is an advantage when there is little information. The work presented in [19] also studies GPU performance prediction. In this case the GPU is used as services in the cloud. The authors use regression techniques to improve the use of resources. Additional studies regarding the prediction of the use of computational resources required by cloud applications can be found in [20].

Performance prediction on large-scale systems such as Web Search Engines (WSE) has been widely studied. Previous work such as [21–24], propose to predict the response time of queries processed on the WSE. The work in [21] uses regression techniques. The authors in [22–24] use machine learning approaches based on the *Fourier Transform*. In all cases, the authors obtain good results, showing that their proposals can be used to properly manage the available hardware resources under unexpected peaks of queries.

Simulation is a useful tool when the complexity of a system prevents its analysis by using analytical methods and has proven to be of great help in predicting performance when the simulators are properly validated and verified [25]. The authors in [26] use Petri nets to simulate a scalable multicore processor infrastructure.

Predicting system performance through computer simulation can be analyzed from three perspectives: simulation of software applications, simulation of user behavior, and simulation of the computing infrastructure on which the applications are running. In the case of simulation of software applications, some studies such as [5,27–29] aim to reflect the behavior of applications compared to the dynamic behavior of users during crisis situations. These studies show that such behavior causes saturation and system crashes. The work in [30], simulates a WSE to compare concurrency control techniques during the query processing process. The work in [31], studies the energy consumption of the WSE, based on the hardware specification, the dominant cost operations and the queries executed by the users. The work in [32] simulates a NoSQL database engine called MongoDB, this allows evaluating the performance of its main operations under a stand-alone or distributed deployment scheme.

The simulation of user behavior includes (1) the mobility models, which describes how users move in an environment [33–43], and (2) how users use the devices like smartphone and how this use influences energy consumption [44–51].

Finally, some simulators have been proposed in the technical literature for the simulation of computing infrastructures. The work presented in [52], studies the behavior of Fat-Tree networks under various configurations for high-scale systems. Similar studies such as, [53,54], developed simulators for wired and wireless network configurations with and without supporting infrastructure. Other works such as [55–58], have implemented simulators to analyze the behavior of computing infrastructure in data centers, cloud and the architecture of the applications that run on them.

3. Computing platform and software applications for natural disasters

This section begins by describing the computing platform used to deploy the software applications designed for post-natural disaster scenarios. We then present two applications as case studies in this work.

3.1. Computing platform

Fig. 1 presents the architecture of the computing platform describing the hardware and software used to deploy the applications. The platform is based on containers. The architecture is devised to run the software applications on commodity hardware usually available from public organizations from Chile. Most components forming the architecture are from existing public domain software, and some of them are consumed as an external service as in the case of Messenger or Telegram.

To deploy the applications on this distributed platform we use the following components: the master, a cache service – it is a type of data repository, which may or may not be persistent – an image repository service, a data repository service, and worker services. The master manages the communication messages with the users of the applications, and between software components. In addition, it is responsible for scaling the application through constant monitoring of the entire platform and raises new application instances if necessary. The master is typically used in container orchestration systems like Kubernetes [59] and Docker-Swarm [60,61]. The image repository holds the images of the containers. That is, the application with all its dependencies without being deployed. The data repository are databases that should support fault tolerance and replication. The cache service keeps the state of the tasks executed in the application. E.g. a volunteer accepts or finishes a task. The worker encapsulates the applications when they are deployed on the platform. That is, the application is undeployed in the image repository and to deploy it, the system set all the environment variables in a configuration file and the deployment is launched using instructions on the orchestration system. Finally, the application is encapsulated in a worker service, which allows the interaction of the application with the rest of the platform.

In Fig. 2, we explain the configuration files utilized for deploying and executing the applications. We show the yaml files used for configuring the actual deployment of the applications, orchestrated by Kubernetes. These yaml files are categorized into two groups: (1) those employed to instantiate the containers (in this case, using Docker) housing the software components of each application (frontend and backend) and (2) the yaml files configuring the external services utilized by the applications, such as the database and cache memory. Furthermore, we provide an example of the yaml configuration file for the database, using MongoDB as a service. This example illustrates the specification of the TCP communication protocol, with the database engine running on port 27017 of the server, which IP address is 192.168.1.104.

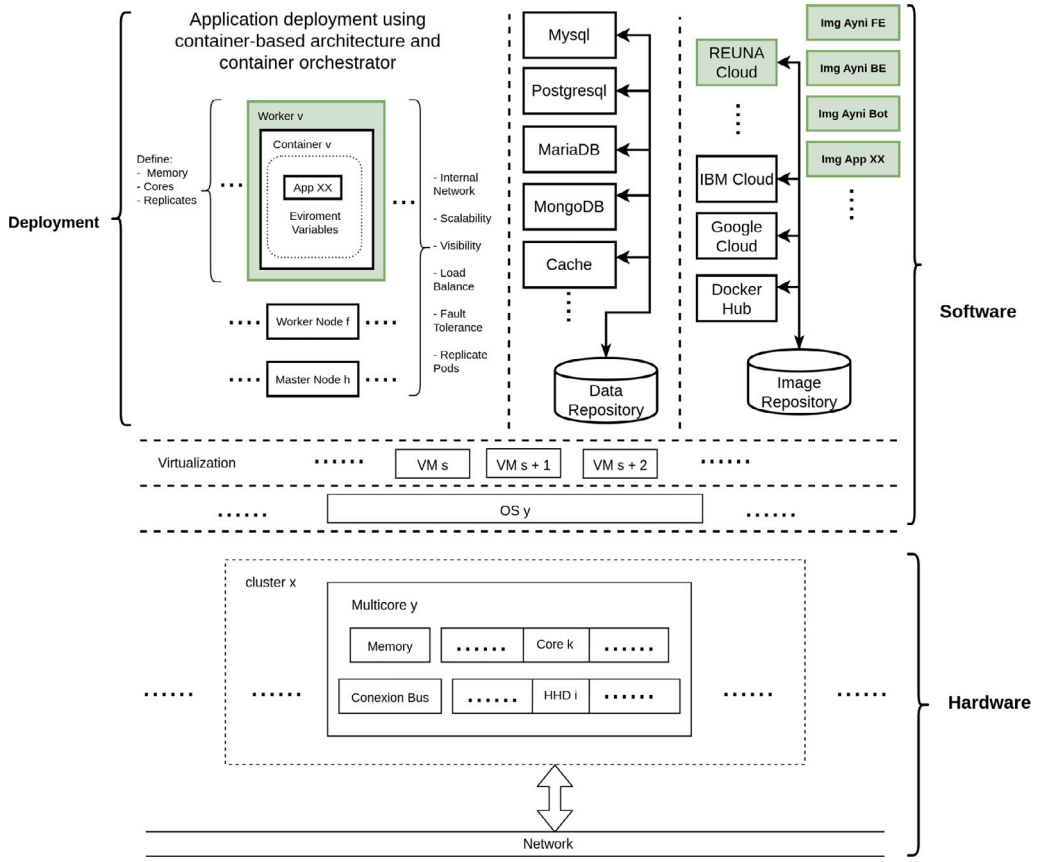


Fig. 1. Architecture of the computing platform. It contains the hardware (computers, network) and the software including the operating system, server virtualization and the applications necessary to deploy the applications based on container technology and container orchestration.

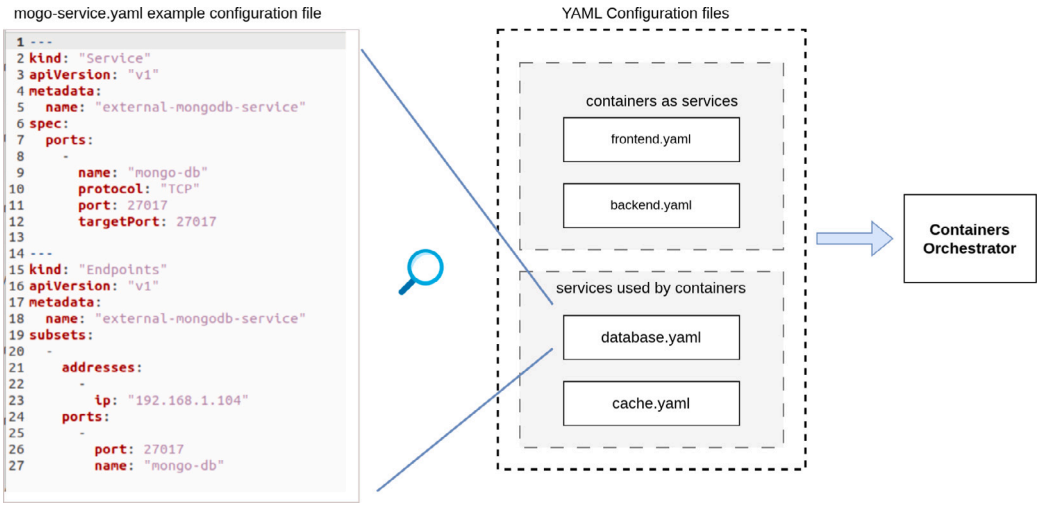


Fig. 2. Yaml configuration files used to deploy the applications on the container orchestrator.

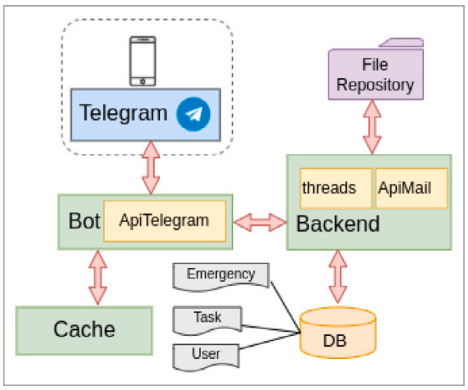


Fig. 3. Services used to implement the software applications.

3.2. Software applications

This section introduces two applications [13] designed to be used after a natural disaster strikes, which serve as case studies in our analysis. These applications have been developed by CITIAPS,[2] in collaboration with expert entities in disaster situations from Chile like SENAPRED,[3] SERNAGEOMIN,[4] MovidosxChile,[5] TECHO[6] and REUNA,[7] who provided the main guidelines for the application designs. The applications are Rimay[8] and Ayni.[9] Both applications have a Telegram Bot.[10]

3.2.1. Rimay

The Rimay application manages and coordinates groups of volunteers using the Telegram bot. Each group of volunteers has a leader who receives tasks from one or more coordinators or administrators. Coordinators can use a web page to easily manage tasks and collect statistics. This application is composed of two backends, a data repository, a file repository and a cache service, which temporarily stores the different states of the actions performed by the user (example: task started and task completed). The first backend called "BOT", which is responsible for bring users to meetings and notify them about the assigned tasks they must complete. The volunteer teams are generally small groups with a leader in charge of managing the tasks to be solved. The second backend manages the interaction between the frontend and the data repositories (database and files). Fig. 3 shows the services used to implement the applications. The Rimay application supports the following tasks:

- Registration: users request to register as volunteers. The data associated with the volunteer is stored in the system. The volunteer can be selected as coordinator of a task or as a normal volunteer.
- Create event: the coordinator creates an event, such as a forest fire. Additionally, the coordinator can create tasks for each event, such as distributing facial masks. Tasks can have information such as where the face masks are stored and a task description. The volunteers can accept or reject tasks.
- Create Task: the coordinator creates one or more tasks for an emergency. The coordinator can finish an event when it is completed.
- Delete User: the coordinator can delete users from the system.
- Send File: The coordinator can send a file to the volunteers.
- Help/Volunteer status: provides help and reports the status of volunteers.
- Send Report: volunteers can send a report to the coordinator.

Rimay also includes other tasks like: Administrator accept volunteer, Cancel a given task, Accept a task and Finish a task.

[2] https://citiaps.usach.cl/
[3] https://senapred.cl/
[4] https://www.sernageomin.cl/
[5] https://www.movidosxchile.cl/
[6] https://cl.techo.org/
[7] https://plaza.reuna.cl/
[8] https://citiaps.usach.cl/portafolio/rimay
[9] https://citiaps.usach.cl/portafolio/ayni
[10] https://telegram.org/

Fig. 4. Steps of the proposed methodology.

3.2.2. Ayni

The Ayni application manages groups of volunteers through a bot of Telegram. These volunteers sign up for and inform their skills (physical, knowledge, medical, etc.), then they go through an automatic selection process. Finally, the volunteer can participate in the tasks created as a result of the emergency. The Ayni application supports the following tasks:

- User Registration: the user request to register as volunteer. The data associated with the volunteer is stored into the system. The user declares his/her skills and conditions requested. For example, you could be asked to indicate whether you have any vaccine.
- Create volunteer: user data is entered into the database through an API.
- Active Emergencies: the specifications of the active emergencies are displayed.
- Volunteers agree to participate: a volunteer agrees to participate in an emergency. From now on, notifications may arrive to participate in tasks related to the emergency. Also, he/she can decline the invitation.
- My Active Emergencies: only active tasks assigned to emergencies are displayed.
- Accept task: a volunteer accepts the task assigned by a leader.

Ayni also includes other tasks like: Create an emergency, Create a task, Help and Cancel.

4. Performance estimation methodology

We propose a methodology to estimate the performance of applications designed to be used after a natural disaster strikes. The methodology has five steps, as shown in Fig. 4: (1) characterize the applications, (2) benchmarks, (3) computing platform model, (4) definition of the resources utilization based on the main application functionalities through descriptive language, and (5) discrete event simulation for performance prediction.

4.1. Characterization of platform applications

In the characterization phase, we define the specific attributes and criteria that applications must satisfy in order to qualify for inclusion within the computing platform. These attributes serve as benchmarks to ensure that the selected applications align with the overarching objectives of the platform and operational standards. Key characteristics encompass various dimensions, including usability, scalability, fault tolerance, elasticity, and mobility.

Usability is paramount, as it dictates the ease with which users can interact with and navigate through the applications. Scalability is essential to accommodate varying workloads and user demands, ensuring that the platform can seamlessly adapt to changing requirements without compromising performance or efficiency.

Furthermore, state-free architecture is favored, as it enables applications to operate independently of their previous states, thereby simplifying deployment and management processes. Fault tolerance is crucial for maintaining system integrity and resilience in the face of potential failures or disruptions, ensuring uninterrupted operation even under adverse conditions.

Additionally, elasticity allows the platform to dynamically scale resources up or down in response to fluctuating demands, optimizing resource utilization and cost-effectiveness. Lastly, mobility features empower applications to traverse the platform effortlessly, facilitating seamless movement across different environments and locations.

By defining and adhering to these comprehensive criteria, we ensure that the applications integrated into the computing platform not only meet the immediate needs of users but also possess the versatility and adaptability to be executed after a natural disaster strikes.

4.2. Performance application benchmarks

The benchmark step involves the following process:

- Code instrumentation: we include instructions in the source code of applications to obtain logs files – which record the processes executed in the applications – the start and end of each task executed when the application is running. Fig. 5 illustrates the instrumentalization of source code and the log file for each component of the application. In this example, the user request is executed in the virtual machine VM1 which runs a frontend component (CFE). When the processing of the request begins (INI), we record the corresponding data into the logeFile1. This process is repeated for each task executed for the request.

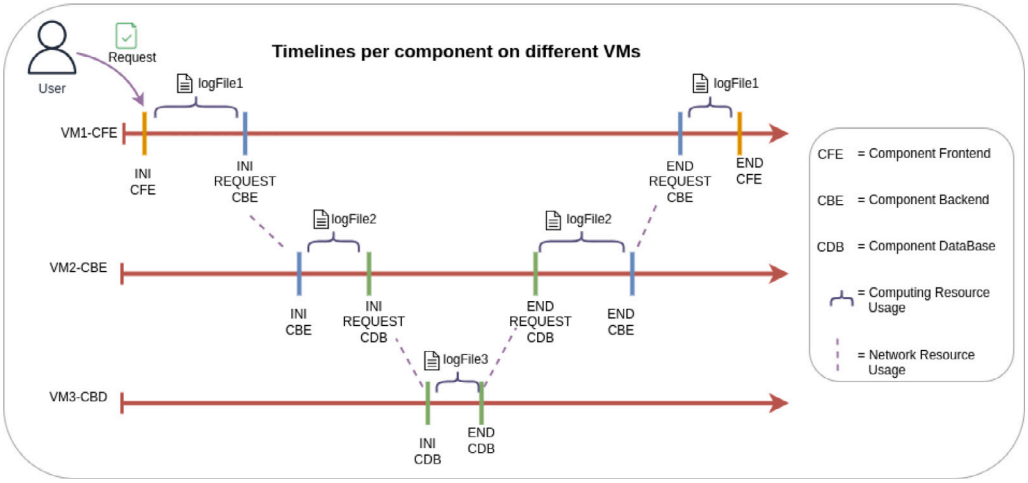


Fig. 5. Timelines for three VMs. The user initiates a request in VM1-CFE. Its information is stored in logFile1. Each MV records the component operations in log files.

- Deployment of the application on the Computing Platform: refers to executing the applications on a real infrastructure. We deploy the logical components of the applications in different Virtual Machines (VMs).
- Run applications: we obtain the workflow of the application by running a set of tests. Each test executes the main functionalities or use cases of the applications such as Registration, Create event, Delete User, etc.
- Log Processing: we analyze the logs retrieved from each test, which provide comprehensive data including the number of cores per server, RAM size and utilization, HDD size and utilization, execution time, CPU utilization, throughput, and network latency.

4.3. Modeling the application platform

In the model application platform step of the proposed methodology, we model the applications that use the computing platform and the computing resources: computers and network inside each cluster and the network connecting the clusters of the distributed platform. The purpose of the computer clusters is to distribute the workload of the applications and guarantee characteristics such as fault tolerance, to deploy a new instance of the applications with similar characteristics in such a way that it does not affect the correct operation of the platform.

We divide the computing platform in GeoZones, that represent the subdivision into geographical zones of the clusters, which communicate through a network infrastructure. Fig. 6 illustrates an example of the GeoZones defined for Chile, including the north, central, and south zones. Within each zone, clusters comprising multicore nodes (MC) are deployed. The communication networks in these zones offer varying bandwidths $AA > BB > CC$. For the network infrastructure we model:

- Connection lines (connLine): it connects the multicores node (MC) inside each cluster, such as fiber optics, wifi, etc. They provide different communication bandwidths, always ordered from greatest to lowest speed, and where the fastest one is always active (the others are kept waiting). If the active line fails, the next one is enabled to provide fault tolerance.
- Network Interconnections (InterConn): it communicates MC from two different clusters. There can be only one possible interconnection, or more than one. In the example of Fig. 6, the option$_1$ uses multiple interconnection links and option$_2$ uses only one interconnection.
- Channel: it is defined as the totality of potential interconnections among the clusters, this facilitates the utilization of alternative paths in the event of failures.

We model the applications as a set of software components or programs and a set of task flows. The task flows represent the logic of the interactions between the components. That is, there will be a complete flow for each use case of the application, which includes the sending of requests between components through the communication network (see Fig. 7). Each component is composed by a set of Services which perform specific action, e.g. in the Rimay application, which has six components (Bot, backend, Cache, DB, Telegram, file repository), we can identify the "Help" service of the "backend". We model the components of the applications using the following elements (Fig. 8):

- Service TimeOut: usually in milliseconds. If this time is exceeded, the component returns an error during the time of execution of the service.
- (IP, Port): the IP identifies the VM where the component is deployed. Port identifies the component deployed within the VM, which may contain a set of components of different applications to better use the available computing resources.

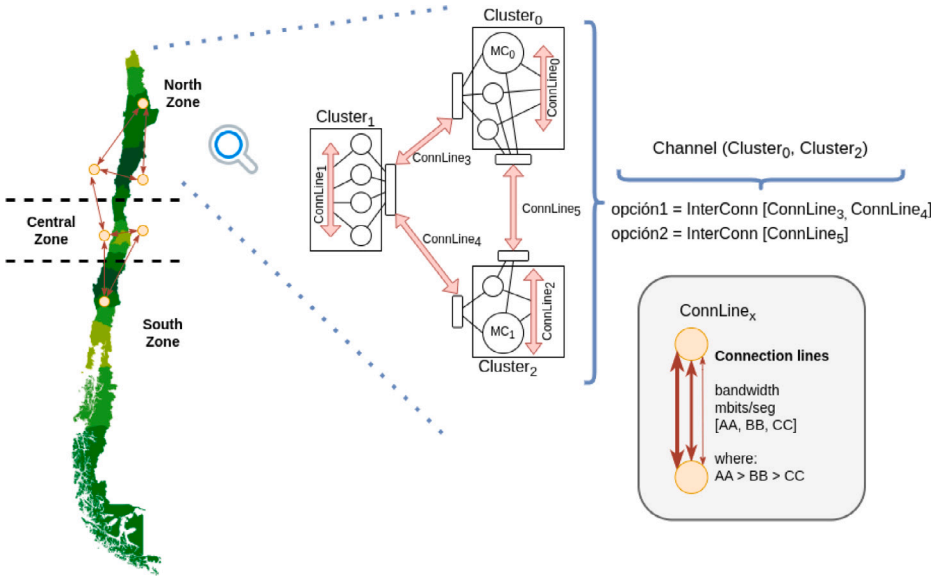


Fig. 6. Computing platform formed by clusters interconnected by a communication network. Furthermore, the subdivision of Chile into geographical zones.

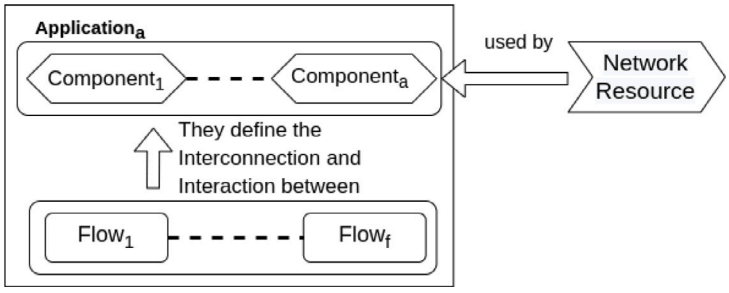


Fig. 7. Definition of an application.

- Deployment Configuration: determines which components are connected.
- Queue: keeps the incoming to the component.
- Current State: describes the three possible states of the component:
  - Ok: the Component is deployed and running.
  - Saturated, for each component we define a threshold for a percentage of use of the resources. If the threshold value is reached, the component can introduce delays in the execution times.
  - Fail: the Component stopped working.
- Services: defines the set of atomic works (internal and/or external) that use the resources of the computing platform and are executed in a single component. In other words, each component has a unique set of associated services.
  - Int_Work (Internal Work): work that must be processed by the component,
  - Ext_Work (External Work): the execution of the work is requested to another component that is part of the application.

The minimum (physical) unit of the computing is a multicore processor (MC), with the cores, RAM memory and HDD. We model each MC as shown in Fig. 9. The function F applied to Work_0, computes the utilization time on the resource r. As the application components, resources have three possible states: Ok, Fail or Saturated.

4.4. Application flows

Before simulating the applications on the platform, we define the task flows representing the sequence of steps for a use case of an application. A task flow is composed by a set of requirements (Req) which define the execution sequence of the services. The task flows use messages to communicate requests from one source component to another: $msg(source, destiny, sizebytes, idService, context)$

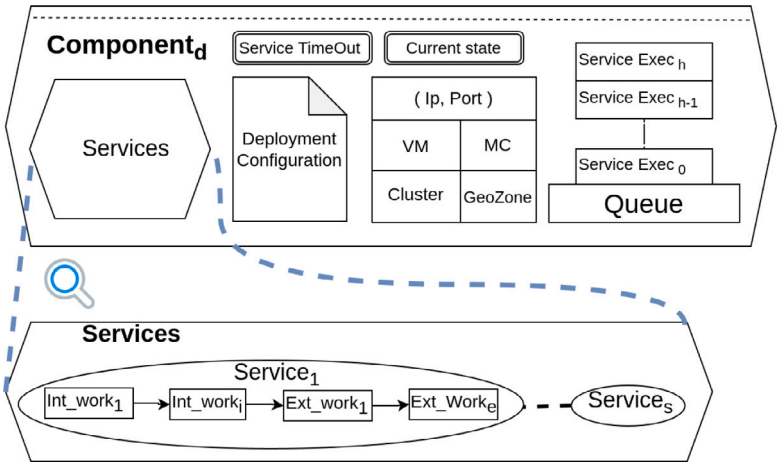


Fig. 8. Definition of the components of the applications.

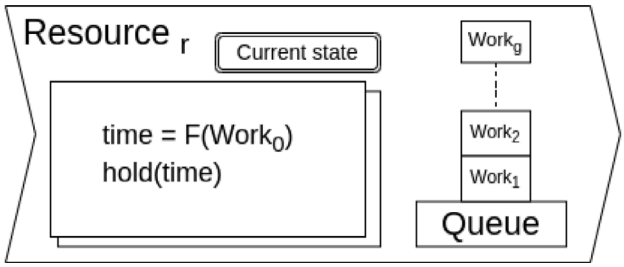


Fig. 9. Model of a computational resource.

the source and destiny are identified using the (Ip, Port). Sizebytes is Size of the message in bytes. IdService is the service to execute in the destiny component. Context has information to process the service, e.g., filter for a search.

Fig. 10 shows how two components communicates through a message msg_e to execute the tasks involved in the Flow$_f$ of Application$_a$. The Application$_a$ has a set of components {Component$_1$, Component$_2$ … Component$_h$ … Component$_a$} and a queue of requests {Request$_1$ … Request$_n$}. To execute the first request, Component$_1$ is linked to Component$_2$ through its Ext_Work$_r$ (source), which generates a service request (msg$_e$). Then, Component$_2$ (destination) executes the Service$_6$.

4.5. Simulation of the applications

Finally, we simulate the deployment of the applications on the distributed platform. We implement the simulator using the LibCppSim library [62]. This library uses coroutines to simulate the concurrent execution of different process. Fig. 11 shows the general classes of our proposes simulator. These classes have sub-classes not included in the figure for a better illustration. The main classes are described below:

- The"Process" class is an specific class of the LibCppSim library which provides the following functions. The *time*() function records the progress of the simulation time. The *hold*() function, simulates time delays and the *passivate*(), deactivates a processes until it is awaken using the function *activate*() or *activate_after*().
- The "Network" class allows to simulate the behavior of the communication network. It includes sub-classes like the Message class, the Channel and ConnectionLine classes, to define the different communication channels and the messages that travel through them. It considers the status of the network (Ok, Saturated, Down) and bandwidth.
- The "PlatformHardware" class contains the main components of the computing infrastructure implemented in the simulator, some of these components are represented by a "Multicore" class, a "Cluster" class and a "VirtualMachine" class. An additional class named "ExecutionRequest", is used to simulate the execution of the requests that the deployed software components send to the Multicore cores to be processed.
- The "PlatformSoftware" class, is used to simulate the universe of deployed applications that are running on the computing infrastructure, and how these are interconnected on the network. The deployed applications are represented by a "ProgramDeployment" class, which is the base class to describe any software component that you want to deploy, since it distributes the workload over the different cores of the Multicore and/or VirtualMachine where it is deployed. It also includes a sub-class

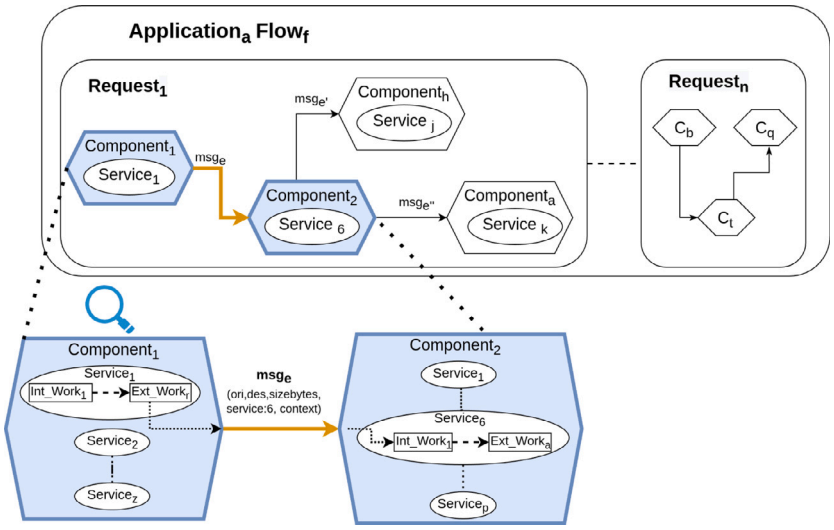


Fig. 10. Representation of a Flow f of an Application$_a$. The Request$_1$ received from the user initiates the execution of a series of jobs executed in the service$_1$ of the component$_1$, in the service$_6$ of the component$_2$, in the service$_j$ of the component$_h$ and in the service$_k$ of component$_a$. The services are composed of internal and external jobs. External jobs generate the sending of messages (msg) between components.

"Work" which is the minimum unit of work that can be executed in the simulator; this class encapsulates the delay value to be used in the simulated cores.

- The "PgmdClient" class is responsible for injecting user requests into the system.
- The "SimTimeLine" class allows different events to be injected throughout the simulation time, and these events cause various situations on the simulated platform. For example: it can be indicated that a Multicore stopped working, therefore, the deployed and running applications have to be replicated in other Multicore to keep the system running.

To simulate the platform and the applications we use json input files. Fig. 12 represents the sets json configuration files, which are organized into two groups: Traces and Computer platform.

- Traces: it consists of two files "User Requests" and "Events resulting from natural disasters". The former contains the set of user requests for each application, while the latter has the events that occur during natural disasters, such as server crashes.
- Computer Platform: it includes four files: "Resources", "Infrastructure", "Applications" and "Application Deployments". The Resource.json file defines the type and number of VMs, specifying the amount of RAM, HDD and cores. The Infrastructure.json file defines the number of multicore processors in each cluster, and the network characteristics. The Application.json file is used to set the characteristics of the applications, the workflows, and the cost function for each work. Finally, ApplicationDeployment.json is used to specify which applications will be deployed on which components of the infrastructure, whether physical (multicore) or logical (VM).

Fig. 13 illustrates an example of the configuration files for "User Requests" and "Events Resulting from Natural Disasters". The "User Requests" file is constructed using traces obtained from piloting the applications during real natural disaster events or evacuation drills. Alternatively, these traces can be generated based on the expertise of individuals experienced in natural disaster situations. The json file contains information such as the message id (idmsg) used to identify the request, the origin (ori) and the destiny (des) of the message, the content of the message which includes information about the application and optional conditions indicating whether a filter or a particular data has to be used. The system flow (sysFlow) contains the sequence of services and components involved in the execution of the request. The file also includes the time (timeHold) the requirement begins its execution or when a particular event occurs, e.g. when a computer crashes. Finally, the "details" element provides more information about the events executed by the simulator, e.g. a server goes down due to a power outage.

In Fig. 14 we detail the four json configuration files for the computer platform. The Resource.json file specify the types of multicore and virtual machines that the platform contains. We classify the VM as large, medium and small, depending on the capacity of each VM (e.g. the number of cores, the RAM memory, etc.). The Infrastructure.json file separates the information into two large sets, the first referring to the multicores where the applications are deployed. It also contains information about the cluster where the multicores are located and the geographical zones where the cluster is located. The second one, gives information about the network that interconnects the different clusters. The Applications.json file, contains the specification of the applications (example: Ayni and Rimay), their software components, the workflows of each software component and the time estimation functions of each job. Finally, the Applicationdeployment.json file, contains information about the types of virtual machines created on each multicore used in the simulated deployment and the specification of the applications deployed on each virtual machine, among others.

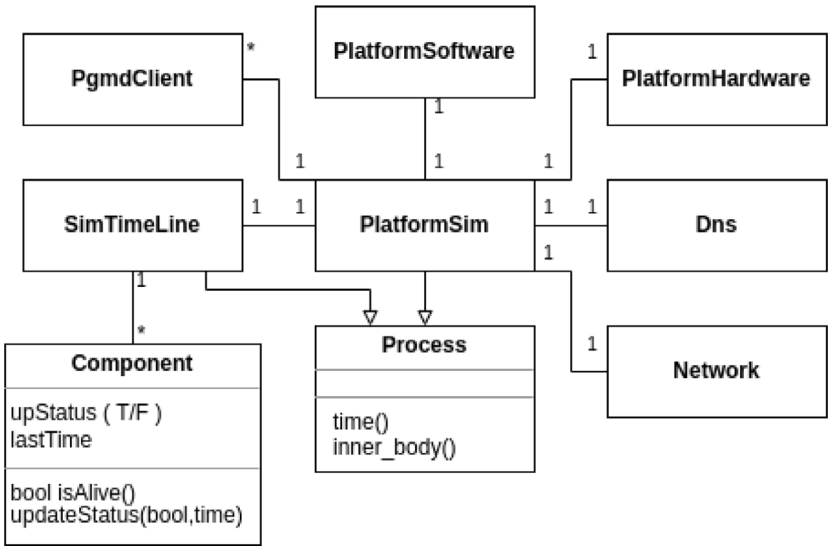


Fig. 11. General class diagram of the simulator.

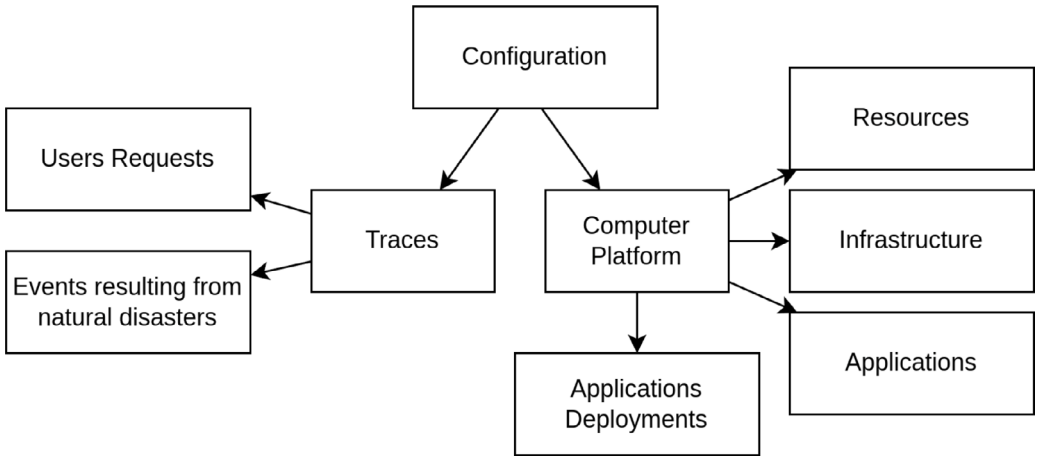


Fig. 12. Json configuration files used to run the simulations.

5. Specification language

Specification languages are fundamental tools in the field of simulation and modeling of dynamic systems. These languages allow for a formal description of system behavior, such as control and communication systems, enabling the analysis of their dynamics. It provides a formal and mathematical specification of system behavior, which facilitates the representation, understanding, and rigorous analysis of the simulated system.

In this work, we introduce a specification language to represent the elements of a computing platform and the communication network. We apply this language to two tasks executed on Ayni and Rimay, but it can be easily applied to similar services. Additionally, this language also allows to estimate the cost of using the platform's resources according to the model presented in the previous section. Fig. 15 shows a summary of the main elements that interact on the platform and are defined with the proposed language.

$$\text{Platform} = \{\text{Resource}_1 \dots \text{Resource}_R\} \cup \{\text{Application}_1 \dots \text{Application}_A\}$$

$R \in \mathbb{Z}^+$ is the different kinds of resource. In this work, we consider the computation resource and the network resources.

$A \in \mathbb{Z}^+$ is the number of applications.

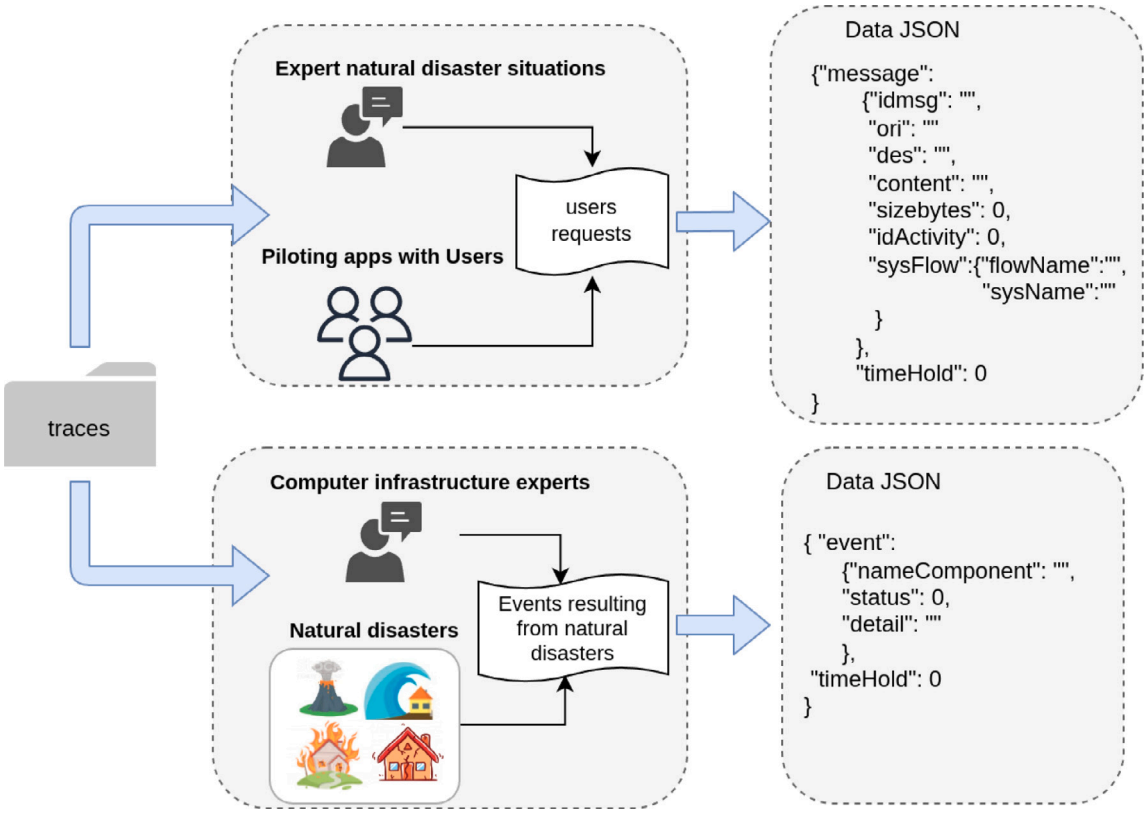


Fig. 13. Json configuration files: traces for user requests and events resulting from a natural disaster.

Therefore, the platform used in this work is specified as follows:

$$\begin{aligned} \text{Platform} \quad = \quad & \{\text{Network, Computational Resources}\} \cup \\ & \{\text{Application}_1 \ldots \text{Application}_A\} \end{aligned}$$

Where

$$\begin{aligned} \text{Network} \quad &= \{\text{Channel}_1 \ldots \text{Channel}_H\} \\ \text{Channel}_h \quad &= \{\text{InterConn}_{h1} \ldots \text{InterConn}_{hI}\} \text{ with h}\in\{1\ldots \text{H}\} \\ \text{InterConn}_{hi} \quad &= \{\text{ConnLine}_{hi1} \ldots \text{ConnLine}_{hiL}\} \text{ with i}\in\{1\ldots \text{I}\} \end{aligned}$$

Additionally, we have:

- $H \in \mathbb{Z}^+$, is the number of communication channels.
- $I \in \mathbb{Z}^+$, is the number of Interconnections (InterConn) available for the channel h.
- $L \in \mathbb{Z}^+$, is the number of connection lines (ConnLine) available in the InterConn i of channel h.

$$\begin{aligned} & ComputationalResource = \{GeoZone_1 \ldots GeoZone_G\} \\ & GeoZone_g = \{Cluster_{g1} \ldots Cluster_{gC}\} \text{ with } g \in \{1 \ldots G\} \\ & Cluster_{gc} = \{MC_{gc1} \ldots MC_{gcM}\} \text{ with } c \in \{1 \ldots C\} \\ & MC_{gcm} = \{Core_{gcm1} \ldots Core_{gcmK}\} \cup RAM_{gcm} \cup HDD_{gcm} \text{ with } m \in \{1 \ldots M\} \\ & Deployment(MC_{gcm}) = \{MV_{gcm1} \ldots VM_{gcmV}\} \\ & VM_{gcmv} = \{Core_{gcm1} \ldots Core_{gcmE}\} \cup RAM_{gcmv} \cup HDD_{gcmv} \text{ with } v \in \{1 \ldots V\} \end{aligned}$$

Additionally, we have:

- $G \in \mathbb{Z}^+$, is the number of geographical zones.
- $C \in \mathbb{Z}^+$, is the number of clusters in the GeoZone g.

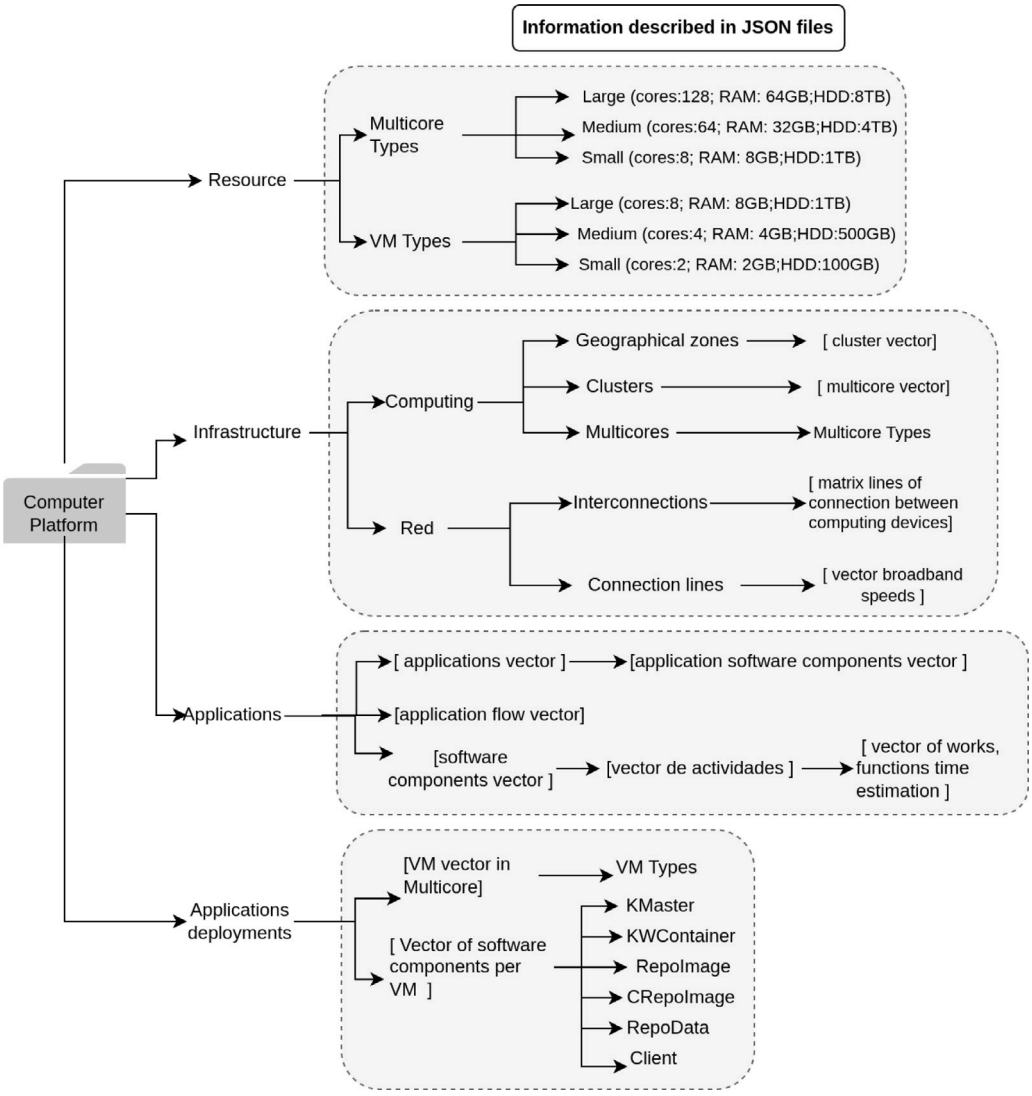


Fig. 14. Json configuration files: specifications related to the computer platform.

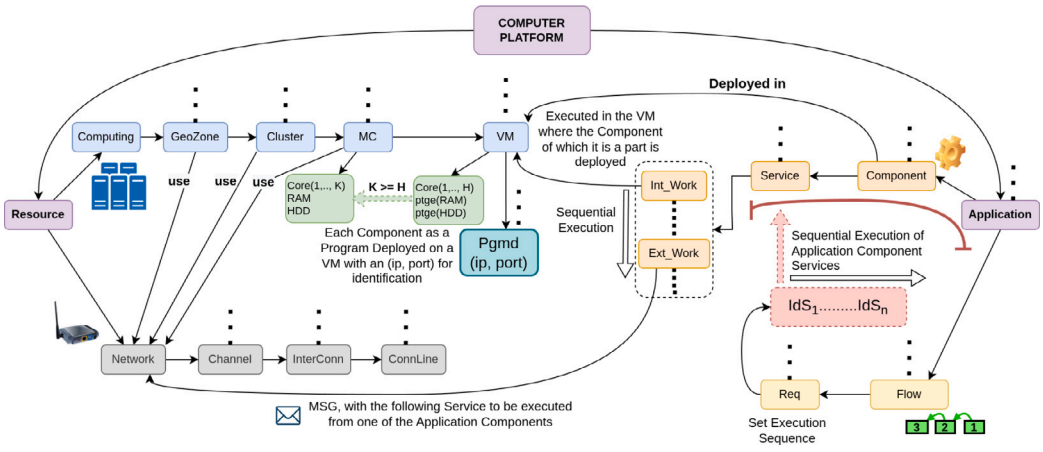


Fig. 15. Main elements of the computing platform and their connections.

- $M \in \mathbb{Z}^+$, is the number of MCs of the cluster c of the GeoZone g.
- $K \in \mathbb{Z}^+$, is the number of cores of the MC m of the cluster c of the GeoZone g.
- In the definition $Deployment(MC_{gcm})$, V is the number of VMs deployed on the MC_{gcm}.
- $K \in \mathbb{Z}^+$, $K \geq E$, where K is the number of cores of the MC_{gcm} and E is the assigned number of cores of the MC_{gcm} to the VM_{gcmv}.
- $RAM_{gcm} \in \mathbb{Z}^+$, is the storage capacity of the RAM memory. Furthermore, $RAM_{gcmv} = Percentage(RAM_{gcm})$, and it holds that $\sum_{v=1}^{V} RAM_{gcmv} \leq RAM_{gcm}$.
- $HDD_{gcm} \in \mathbb{Z}^+$ is the disk storage capacity. Furthermore, $HDD_{gcmv} = Percentage(HDD_{gcm})$, and it holds that $\sum_{v=1}^{V} HDD_{gcmv} \leq HDD_{gcm}$.

$Application_a = \{Component_{a1} \dots Component_{aCO}\} \cup \{Flow_{a1} \dots Flow_{aF}\}$ where

- $\forall a \in \{1 \dots A\} \in \mathbb{Z}^+$, with a as the application identifier within the platform, and A is the amount of applications.
- $CO \in \mathbb{Z}^+$, is the number of application Components_a
- $F \in \mathbb{Z}^+$, is the number of flows of the Application_a

$Component_{aco} = \{Service_{aco1} \dots Service_{acoS}\}$ where $co \in \{1 \dots CO\}$
Additionally, we have:

- $S \in \mathbb{Z}^+$, is the number of services that can be executed in the Component_{aco} of the Application_a.
- Each Component_{aco} is deployed on a VM. Additionally, we can send messages with execution requests to other application Components_a.

$Service_{acos} = \{Work_{acos1} \dots Work_{acosT}\}$ where $s \in \{1 \dots S\}$
Additionally:

- $T \in \mathbb{Z}^+$, is the number of ***works*** that must be executed in the Service_{acos} of the Component_{aco} of the Application_a.
- Work_{acost}, where $t \in \{1 \dots T\}$, is the minimum execution unit of the platform and is estimated by a cost function that depends on the Resource_r. The Work_{acost} can be:
  - Internal Work: it is executed in the Component_{aco}, and is processed in a VM that is part of the computing resource.
  - External Work: executed on a Component_{ab}, where $b \in \{1 \dots CO\}$ and $b \neq co$. That is, Component_{ab} is different from the one used by $\text{Work}_{acos(t-1)}$ in Service_{acos} of Component_{aco}. To this end, a message $(source, destination, sizebytes, idService, context)$ is sent to the Component_{ab}.

We define FCC as the function used to estimate the costs of works. It represents the use of a core, RAM or HDD when executing the internal work t of the service s of the component co of the application a in a VM v.

$$\begin{aligned} \text{FCC}(\text{Model}_a; \text{VM}_{gcmv}, \text{Work}_{acost}) \quad = \quad & \text{SimulaCost}(\text{CoreState}(), \\ & \text{RAMState}(), \\ & \text{HDDState}(), \\ & \text{ComponentState}()) \end{aligned}$$

- Model_a, is a prediction model used for application a. Each internal work executed by the application is represented by models based on machine learning or models based on discrete or continuous statistical distribution, among others.
- VM_{gcmv} is the deployment of the Component_{aco} on a VM of the platform.
- Work_{acost}, is the work that must be executed in the Component_{aco}.
- CoreState(), estimates the utilization percentage of the core.
- RAMState(), utilization of the RAM memory in GB in each MC of the computing platform.
- HDDState(), utilization percentage of HDD in GB.
- ComponentState(), it retrieves the values OK, Saturated or Fail.

We also define FCR as the network cost function. It is used to estimate the network utilization.

$$\begin{aligned} \text{FCR}(\text{Component}_{am}; \text{Component}_{an}; \text{msg}) \quad = \quad & \text{SimulaCost}(\text{msg size}, \\ & \text{communication bandwidth} \\ & \text{ChannelState}()) \end{aligned}$$

- Component_{am}, is the source component of the message.
- Component_{an}, is the destination component of the message.
- msg, message from Component_{am} to Component_{an} with information of the service to be executed.
- Size of the message communicated between Component_{am} and Component_{an}.
- Communication bandwidth of the connection lines (connLines) used by the communication channel (Channel) to transfer the message.

- ChannelState(), retrieves the state of the channel (OK, fail or saturated).

$$\text{Flow}_{af} \quad = \quad \{Req_{af1} \dots Req_{afr} \dots Req_{afR}\} \text{ where } f \in \{1 \dots F\}$$

Additionally, we have:

- $\forall r \in \{1 \dots R\} \in \mathbb{Z}^+$, r is a particular request Req and R represents the total amount of Req of the Flow_{af}. All $Reqs$ of a flow are executed sequentially and in order from 1 to R.

$$\begin{aligned} \text{Req}_{afr} \quad = \quad & \{\text{serviceid}_{afr1} \dots \text{serviceid}_{afrq} \dots \text{serviceid}_{afrQ}\} \\ & \text{where } q \wedge Q \in \{1 \dots \max\{S\}\}, \text{ S is the number of component} \\ & \text{services that are part of Application}_a. \end{aligned}$$

where:

- serviceid_{afrq}, is the identifier of a particular service, which can be mapped through a hash function to the Service_{acos} of the Component_{aco} of the Application_a.
- $\forall q \in \{1 \dots Q\} \in \mathbb{Z}^+$, where $q < Q$ is the order of the sequential execution of all services of the Req_{afr}.

Based on the above, the Req_{afr} can be defined as follows:

$$\begin{aligned} \text{Req}_{afr} \quad &= \quad \{ \text{serviceid}_{afr1} \dots \text{serviceid}_{afrQ} \} \\ \text{Req}_{afr} \quad &= \quad \{ \text{hash}(\text{serviceid}_{afr1}) \dots \text{hash}(\text{serviceid}_{afrQ}) \} \end{aligned}$$

$$\begin{aligned} \text{Assuming that} \quad & \text{hash}(\text{serviceid}_{afr1}) = \text{Service}_{ax1} \text{ and} \\ & \text{hash}(\text{serviceid}_{afrQ}) = \text{Service}_{adQ} \\ & \text{where } x \wedge d \in \{1 \dots CO\} \text{ for Application}_a. \end{aligned}$$

Then, we have:

$$\begin{aligned} \text{Req}_{afr} \quad &= \quad \{ \text{Service}_{ax1} \dots \text{Service}_{adQ} \} \\ \text{Req}_{afr} \quad &= \quad \{ \{\text{Work}_{ax11} \dots \text{Work}_{ax1G} \} \dots \{\text{Work}_{adQ1} \dots \text{Work}_{adQN} \} \} \\ \text{Req}_{afr} \quad &= \quad \{ \text{Work}_{ax11} \dots \text{Work}_{ax1G} \dots \text{Work}_{adQ1} \dots \text{Work}_{adQN} \} \end{aligned}$$

where:

- $\{ax11 \dots ax1G\}$ is the sequential execution order of Service_{ax1}.
- $\{adQ1 \dots adQN\}$ is the order of the sequential execution of the Service Work_{adQ}.

5.1. Specification language for Ayni

In this section we show how to use the specification language for the task "User Registration" of Ayni. Other task flows can be obtained using this language in a similar way. In Fig. 16 we show all the components and the main tasks executed in the flow "User Registration" of the Ayni application. The flow consists of two requests (Req01 and Req02), which are divided into two parts. At the top, we show the Orchestrator and its KMaster component, responsible for receiving and redirecting messages to the Ayni application. At the bottom, we show the components of the Ayni application and the main tasks.

During the first requirement (Req01), the Ayni client requests to register into the application using the bot component (Work-Bot), which searches in the cache using the Redis component if it has any previous request (Get-CACHE), and then load the current instruction into cache (Put-CACHE). Then, the bot requests information about the client to the backend (Work-Backend), which in turn sends the request to the MongoDB database (Find-DB). Finally, the bot responds to the client through Telegram (Api-Telegram) with the options "Accept" or "Cancel".

In the request (Req02), the client "Accepts" the registration, this message reaches the bot (Work-Bot), who searches the cache (Redis) if it has any previous request (Get-CACHE), and then load the current instruction into cache (Put-CACHE). Then, the bot requests information about the client to the backend (Work-Backend), which in turn sends the request to the MongoDB database (Find-DB). Then, the bot stores the information of the client, by sending a request to the backend (Work-Backend), which in turn sends the request to the MongoDB database (Create-DB). Finally, the bot deletes (Remove-CACHE) from the cache (Redis) any instructions related to the client, and through Telegram (Api-Telegram) notifies the client.

Next, we apply the specification language for the two requests Req01 and Req02.

$$\begin{aligned} \text{Application}_{Ayni} \quad = \quad & \{\text{Bot, Backend, Redis, MongoDB, Telegram}\} \cup \\ & \{\text{A06, } \dots\}, \text{ where A06 corresponds to the flow} \\ & \text{"User Registration"} \end{aligned}$$

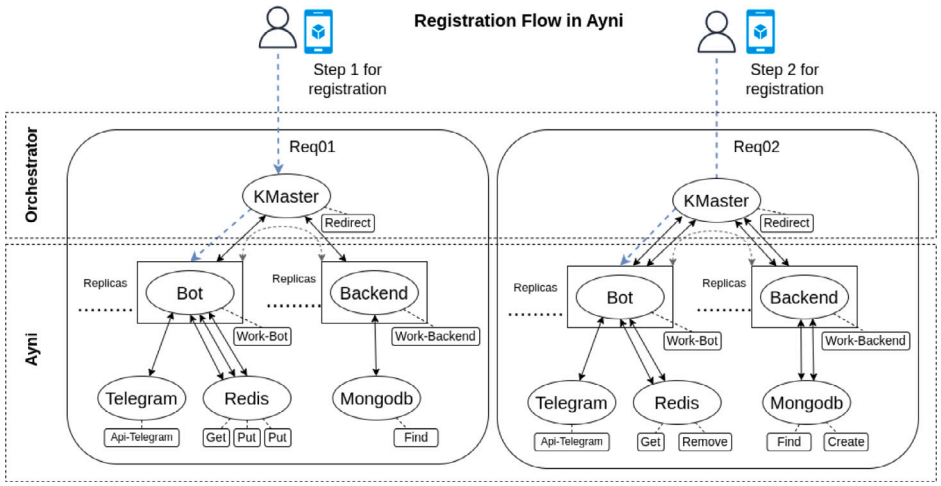


Fig. 16. Requirements executed in the flow "User Registration" of the Ayni application.

$\text{Component}_{Ayni,Bot}$ = { User_Registration, Task_details, Help, … }
$\text{Component}_{Ayni,Backend}$ = { User_Registration, Task_details, Help, … }
$\text{Component}_{Ayni,Redis}$ = {Get-CACHE,Put-CACHE, ContainsKey-CACHE }
$\text{Component}_{Ayni,MongoDB}$ = { Create-DB, Find-DB, List-DB, Update-DB }
$\text{Component}_{Ayni,Telegram}$ = { Api-Telegram }

$\text{Service}_{Ayni,Bot,User\_Registration}$ = {Work-Bot}
$\text{Service}_{Ayni,Backend,User\_Registration}$ = {Work-Backend}
$\text{Service}_{Ayni,Redis,Get-CACHE}$ = {Get-CACHE}
$\text{Service}_{Ayni,Redis,Put-CACHE}$ = {Put-CACHE}
$\text{Service}_{Ayni,Redis,Remove-CACHE}$ = {Remove-CACHE}
$\text{Service}_{Ayni,MongoDB,Find-DB}$ = {Find-DB}
$\text{Service}_{Ayni,MongoDB,Create-DB}$ = {Create-DB}
$\text{Service}_{Ayni,Telegram,Api-Telegram}$ = {Api-Telegram}
$\text{Flow}_{Ayni,A06}$ = {Req01, Req02}
$\text{Flow}_{Ayni,A06,Req01}$ ={idS_1,idS_3, idS_4, idS_2, idS_6, idS_4, idS_8}

Where:
$hash(idS_1) = \text{Service}_{Ayni,Bot,User\_Registration}$
$hash(idS_2) = \text{Service}_{Ayni,Backend,User\_Registration}$
$hash(idS_3) = \text{Service}_{Ayni,Redis,Get-CACHE}$
$hash(idS_4) = \text{Service}_{Ayni,Redis,Put-CACHE}$
$hash(idS_6) = \text{Service}_{Ayni,MongoDB,Find-DB}$
$hash(idS_8) = \text{Service}_{Ayni,Telegram,Api-Telegram}$

$\text{Flow}_{Ayni,A06,Req02}$ ={idS_1, idS_3, idS_2, idS_6, idS_2, idS_7,idS_5, idS_8 }
Where, in addition to the previous definitions, we have:
$hash(idS_5) = \text{Service}_{Ayni,Redis,Remove-CACHE}$
$hash(idS_7) = \text{Service}_{Ayni,MongoDB,Create-DB}$

Then, the computation and network cost functions are applied as follows. Assuming the model Model_{Ayni} and that the components are deployed in the VMs: VM_{gcma}, VM_{gcmb}, VM_{gcmc}, VM_{gcmd} and VM_{gcmz}, then we have:

$\text{Flow}_{Ayni,A06}$ = {
Req01 {
FCC(Model_{Ayni},VM_{gcma}, $\text{Service}_{Ayni,Bot,User\_Registration}$[Work-Bot]),{
FCR($\text{Service}_{Ayni,Bot}$, $\text{Service}_{Ayni,Redis}$, msg) {
FCC(Model_{Ayni},VM_{gcmb},
$\text{Service}_{Ayni,Redis,Get-CACHE}$[Get-CACHE])

```
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Redis}, msg) {
              FCC(Model_{Ayni},VM_{gcmb},
              Service_{Ayni,Redis,Put-CACHE}[Put-CACHE])
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Backend}, msg) {
              FCC(Model_{Ayni},VM_{gcmc},
              Service_{Ayni,Backend,User_Registration}[Work-Backend]),
              FCR(Service_{Ayni,Backend}, Service_{Ayni,MongoDB}, msg) {
                FCC(Model_{Ayni},VM_{gcmd},
                Service_{Ayni,MongoDB,Find-DB}[Find-DB])
              },
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Redis}, msg) {
              FCC(Model_{Ayni},VM_{gcmb},
              Service_{Ayni,Redis,Put-CACHE}[Put-CACHE])
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Telegram}, msg) {
              FCC(Model_{Ayni},VM_{gcmz},
              Service_{Ayni,Telegram,Api-Telegram}[Api-Telegram])
            }
        }},

    Req02 {
    FCC(Model_{Ayni},VM_{gcma}, Service_{Ayni,Bot,User_Registration}[Work-Bot]),{
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Redis}, msg) {
              FCC(Model_{Ayni},VM_{gcmb},
              Service_{Ayni,Redis,Get-CACHE}[Get-CACHE])
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Backend}, msg) {
              FCC(Model_{Ayni},VM_{gcmc},
              Service_{Ayni,Backend,User_Registration}[Work-Backend]),

              FCR(Service_{Ayni,Backend}, Service_{Ayni,MongoDB}, msg) {
                FCC(Model_{Ayni},VM_{gcmd},
                Service_{Ayni,MongoDB,Find-DB}[Find-DB])
              },
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Backend}, msg) {
              FCC(Model_{Ayni},VM_{gcmc},
              Service_{Ayni,Backend,User_Registration}[Work-Backend]),
              FCR(Service_{Ayni,Backend}, Service_{Ayni,MongoDB}, msg) {
                FCC(Model_{Ayni},VM_{gcmd},
              Service_{Ayni,MongoDB,Create-DB}[Create-DB])
              },
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Redis}, msg) {
              FCC(Model_{Ayni},VM_{gcmb},
              Service_{Ayni,Redis,Remove-CACHE}[Remove-CACHE])
            },
            FCR(Service_{Ayni,Bot}, Service_{Ayni,Telegram}, msg) {
              FCC(Model_{Ayni},VM_{gcmz},
              Service_{Ayni,Telegram,Api-Telegram}[Api-Telegram])
            }
        }
    }
```

Fig. 17 illustrates the relationship between the components, services, and computing resources in the distributed platform for the Ayni application and the specification language. This figure includes five components of the Ayni application: Bot, Redis, Backend, MongoDB and Telegram. Each component is deployed on different VMs which use computing resources and network resources, depending on the services executed by the users. For example, the first step of the requirement (Req01) of the "User Registration" task involves accessing the Work-Bot component, with the FCC function used to estimate the cost of this operation. Subsequently,

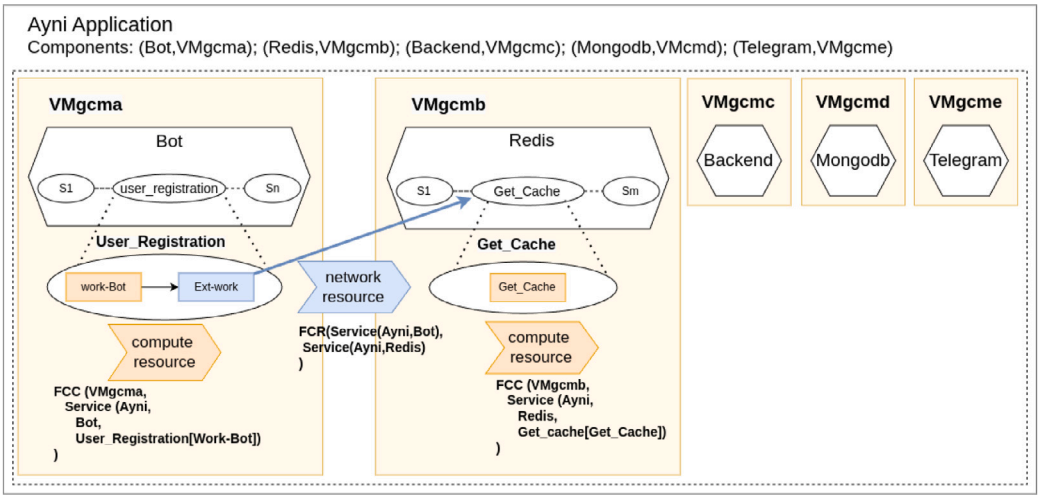


Fig. 17. Representation of the Ayni application deployed on a set of Virtual Machines, the resources and their relationship with the specification language.

the Bot component requests an external work to the Redis component to execute the Get-Cache internal work. The FCR function is used to estimate the network cost. Within the Redis component, the Get-Cache internal work utilizes computational resources, with its cost estimated using the FCC function.

6. Experiments

The experiments are designed to validate and test the proposed methodology, ensuring its accuracy and reliability in predicting the performance of software applications used after a natural disaster strikes. The validation involves assessing whether the methodology accurately predicts the performance of the software applications. In addition to validation, we apply the methodology to assess the performance of applications in response to specific natural disasters, such as a tornado and a wildfire. The application of the methodology to these cases provides insights into its practical utility and effectiveness in diverse emergency scenarios. The experiments involve setting up simulated scenarios, including data configurations and the computing platform, to test the predictive capability of the methodology. By modeling and simulating different components on a distributed computing platform, the methodology predicts the performance outcomes for applications in emergency situations.

6.1. Experimental settings

We compare the results reported by our proposed methodology and the results achieved by the simulator implemented with library LibCppSim [62] against a real implementation of the applications. To deploy the real applications we encapsulate the application components in Dockers [63]. Kubernetes [59] is used to achieve platform orchestration. A Master program manages the Dockers which are encapsulated in Workers. The Docker-registry is used for the image repository,[11] which is part of the container orchestration using Kubernetes. Yaml is used to create the platform configuration files[12] to deploy Ayni and Rimay on the real computing platform. Finally, we use Redis [64] for the cache and we submit the test to the platform using Jmeter [65].

We executed Ayni and Rimay on five multicores, all operating on Ubuntu Server 18.04. Three of these multicores are equipped with 8 cores, 32 GB of RAM, 1.2TB HDD, and a CPU running at 1400MHz. The remaining two multicores have 32 cores, 1.8TB HDD, and a CPU operating at 1298MHz. One of these has 64 GB of RAM, while the other is configured with 128 GB. The data transfer speed for all systems is 1 Gbits. To isolate the measurements of each application component, we use server virtualization, as shown in Table 1, where X={1,2,3} indicates the number of cores per VM. We use two workers so that Kubernetes can distribute the workload.

We include additional instructions in the codes of the components of both applications at the beginning and at the end point of each atomic work to run the benchmarks and to measure the utilization of the resources. Fig. 18 shows the network traffic, and Fig. 19 shows the cores, RAM and HDD utilization. In both figures the information is collected in intervals of 1 second. Similarly, we can obtain the benchmark for other components of the applications.

[11] https://kubernetes.io/docs/concepts/containers/images/
[12] https://yaml.org/

Table 1
Hardware configuration for the experiments.

VM	Component	RAM	HDD	Cores
repoimage01	Container image repository	4	100	2
repodata01	Database	4	100	2
cache01	cache memory	4	100	2
Master01	Kubernetes Master	4	100	2
Worker01	Kubernetes worker	4	100	X
Worker02	Kubernetes worker	4	100	X

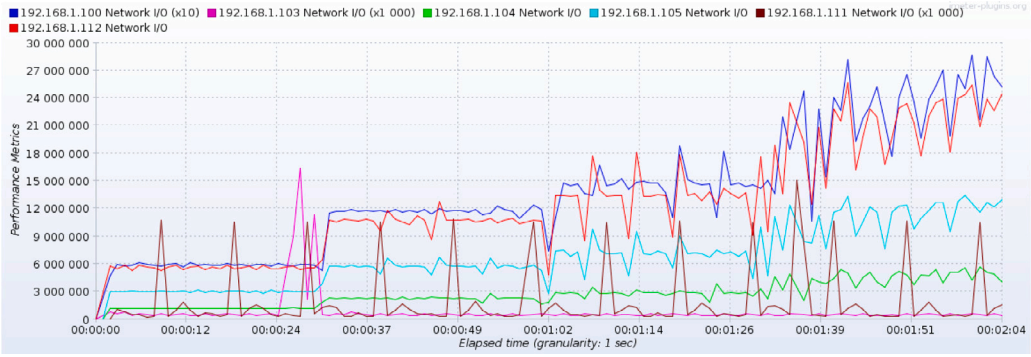


Fig. 18. Network utilization (bits per second) reported by the Ayni application.

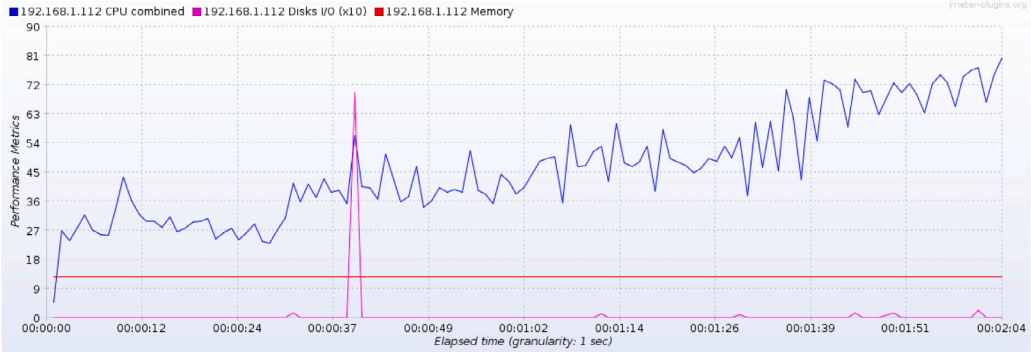


Fig. 19. Utilization percentage of the cores, RAM and HDD of the backend component of the Ayni application.

6.2. Validation

In this section we validate the results obtained by our proposal. To this end, we compare the results obtained when executing the applications Ayni and Rimay on the multicores and the results obtained with the simulator. Table 2 presents a summary of the 21 task flows (11 from Ayni and 10 from Rimay) and their identifiers (ID). We also include, the number of interactions between the client and the application to complete each flow, highlighted in the Requests column. Finally, we show the total number of atomic works (#Works) executed by each flow and the average execution time of each flow in milliseconds.

To estimate the cost of the atomic computing works (function FCC) executed by each flow we use the following techniques: Simple Linear Regression (SLR), Multiple Linear Regression (MLR), Polynomial Regression (PR), and an Ad-hoc technique which is based on a discrete empirical distribution build with data obtained with the benchmarks. These techniques are trained with a dataset constructed in the following manner. The estimated costs are used in the simulator by the Work sub-class of PlatformSoftware class. For each task flow in the applications, we run 15.000 tests. After obtaining the results, we eliminate outliers using the Interquartile Range technique [66]. This led to a 10% reduction in the dataset.

The SLR provides a way to model and understand the relationship between two variables by fitting a straight line to the observed data. It generates a straight line ($y = ax + b$) that allows predicting the behavior of one of the variables (y) based on the other (x). a is the slope of the line, and b is the y-intercept (the value of y when x is 0). In this work, we apply the SLR to predict the time in milliseconds it takes to execute an atomic job ($y = msTime$) by using the size of the message ($x = msgsize$).

The MLR assumes a linear relationship between the dependent variable (y) and multiple independent variables (x). It creates a linear prediction model to compute the output $y = msTime$, based on a set of independent variables. In this case we

Table 2
Requests and atomic works for each flow executed in the applications.

App.	ID	Flow	Req.	#Works	Time
	A01	01 Create Emergency	1	2	79,4
	A02	02 Create Task	1	2	69,05
	A03	03 Create Volunteer	1	2	100,45
	A04	04 Help	1	7	397,06
	A05	05 Cancel	1	8	1112,91
	A06	06 User Registration	2	15	503,36
	A07	07 Decline invitation	2	18	532,1
	A08	08 Active Emergencies	4	47	428,93
	A09	09 My Active Emergencies	4	46	703,98
	A10	10 My tasks	2	22	770,83
Ayni	A11	11 Accept Task	1	2	46,85
	R01	00 Registration	3	37	24,2
	R02	02 Help	1	9	5,45
	R03	03 Create Event	3	69	17,71
	R04	04 Create task	7	131	48,16
	R05	05 Volunteer state	1	21	4,98
	R06	06 Cancel	1	8	5,01
	R07	07 Accept task	4	79	31,28
	R08	08 Finish task	4	75	60,75
Rimay	R09	09 Send file	7	76	387,62

Table 3
Pearson correlation and RMSE obtained for each task flow of the Ayni application.

Flow Ayni	Pearson				RMSE			
	AdHoc	SLR	MLR	PR	AdHoc	SLR	MLR	PR
A01	0,99	0,94	0,94	0,98	7,26	11,77	10,91	13,18
A02	0,99	0,94	0,94	0,99	6,29	13,59	12,05	15,01
A03	0,99	0,94	0,94	0,99	6,63	12,89	11,77	11,83
A04	0,99	0,92	0,92	0,97	7,77	23,05	25,12	16,49
A05	0,99	0,92	0,92	0,98	5,07	22,19	22,94	18,96
A06	0,99	0,96	0,96	0,99	2,60	7,69	8,15	3,79
A07	0,99	0,97	0,97	0,99	2,21	6,17	7,39	5,14
A08	0,99	0,95	0,95	0,98	2,70	19,75	19,44	21,03
A09	0,97	0,95	0,95	0,98	7,73	19,84	20,87	20,50
A10	0,99	0,96	0,96	0,98	3,74	25,97	27,09	26,15
A11	0,99	0,94	0,94	0,99	4,82	11,90	11,07	11,03
Average	**0,99**	**0,95**	**0,95**	**0,98**	**5,16**	**15,89**	**16,07**	**14,83**

use the following independent variables forming a vector $Basevector = \{MC_{Type}, Application_{id}, Component_{id}, Service_{id}, Work_{id}, flgRam, flgHDD, flgCore, flgSat, Core_{id}, msgsize, msgContext\}$, where MC_{Type} is the type (large, medium or small according to the amount of RAM, HDD, and cores) of MC where the VM are created, $Application_{id}$ is the application identifier, $Component_{id}$ is the component identifier, $Service_{id}$ is the service identifier, $flgRam$, $flgHDD$ and $flgCore$ indicates whether the corresponding resources are used. $msgsize$ is the size of the message been processed, and $context$ has information to process the request. The PR, models the relationship between the independent variable ($x = Basevector$) and the dependent variable ($y = msTime$) as an nth-degree polynomial. In simple terms, it involves fitting a polynomial equation to the data instead of a straight line (as in simple linear regression) to capture more complex relationships.

The Ad-hoc technique emerges from a statistical analysis of execution time data, with the primary objective being the computation of the $msTime$ value based on statistical distributions. To achieve this, the dataset is partitioned into N smaller sets to derive the probability distribution for each of these subsets. We experimentally found that $N = 3$ allows to achieve good results.

Tables 3 and 4 show the Pearson correlation and the Root Mean Square Error (RMSE) for the execution times estimated with our proposed methodology and the times obtained with a real execution of the applications. We compute the RMSE as $RMSE = \sqrt{\frac{1}{N} * \sum_i^N (y_r - y_s)^2}$ where y_r, is the real value, y_s, is the valued obtained with the simulation, $\bar{y_r}$ is the average of the real values obtained and N is the sample size.

Results show that the Pearson correlation is always close to one. That is, there is a strong agreement between the real values and the simulated ones. Furthermore, the lowest RMSE is always obtained with the Ad-hoc technique, which reports errors between 2,2% and 7,7% for Ayni, and between 1,7% and 5,5% for Rimay. In case of the Ayni application, the maximum error is reported for the A10 flow, with values between 25,9% and 27,0%. In the Rimay application, the flow R05 reports the highest errors with values between 20% and 21,6%.

Fig. 20(a) presents the "LIST-MONGODB" work of the "Active Emergencies" flow of the Ayni application. It performs a search operation with a filter on the MongoDB database, and 20(b) presents the "CREATE-MONGODB" work of the "Create Emergency" flow of the Ayni application. It executes an insert on the MongoDB database. The results are obtained with the techniques based on

Table 4
Pearson correlation and RMSE obtained for each task flow of the Rimay application.

Flow Rimay	Pearson				RMSE			
	AdHoc	SLR	MLR	PR	AdHoc	SLR	MLR	PR
R00	0,99	0,97	0,95	0,99	2,66	10,96	10,76	9,59
R01	0,99	0,96	0,96	0,99	4,35	18,61	19,86	21,42
R02	0,99	0,97	0,97	0,99	4,03	6,81	7,88	7,001
R03	0,99	0,95	0,95	0,99	3,97	17,02	18,97	17,97
R04	0,98	0,95	0,95	0,99	5,51	15,36	17,71	17,91
R05	0,99	0,95	0,95	0,98	4,64	20,04	21,86	21,65
R06	0,99	0,97	0,96	0,99	3,37	14,35	16,0	14,99
R07	0,99	0,95	0,97	0,99	1,78	9,17	9,73	8,83
R08	0,99	0,95	0,95	0,99	2,68	9,38	9,88	8,84
R09	0,99	0,96	0,97	0,99	2,73	7,93	8,86	6,89
Average	**0,99**	**0,96**	**0,96**	**0,99**	**3,57**	**12,96**	**14,15**	**13,51**

Table 5
Results obtained with Rimay for the tornado of 2019 and results estimated with our simulator.

Flow Rimay	Tests	Time(ms)				
		Real	SLR	MLR	PR	Ad-hoc
R01	294	77,57	70,90	71,04	70,14	76,29
R02	1177	11,80	12,24	12,42	12,54	11,95
R03	465	148,90	169,54	170,53	173,40	148,99
R05	775	16,60	19,40	19,69	19,85	16,71
R07	310	124,49	128,71	129,14	131,02	124,17
R08	260	116,39	119,11	119,78	121,77	117,47

linear regression SLR, MLR, PR and also with the Ad-hoc. We also show results with a Neuronal Network (NN) which are designed to process information and learn patterns through interconnected nodes (neurons) organized in layers and recognize patterns in data. In this work we set the NN with 3 layers and 3 neurons per layer. The Support Vector Machine (SVM) is a machine learning technique that classifies data points by finding the optimal hyperplane in a high-dimensional space, maximizing the margin between different classes. It uses the parameters C and ϵ which are related to the error tolerance. C controls the penalty for errors, and ϵ sets the tolerance for acceptable predictions. In this work set C=4 and ϵ=0,02. We also apply the Random Forest which is an ensemble learning method that builds a multitude of decision trees during training and outputs the mode of the classes (classification) or the average prediction (regression) of the individual trees. It is less prone to overfitting compared to individual decision trees. It uses the maxDepth parameter to determine the maximum depth allowed for each decision tree and the randomstate parameter to initialize the random number generator. In this work we set maxDepth=4 and randomstate=5. The parameters for the machine learning techniques were chosen based on their performance in experiments, where various values were tested. Results show that the no linear regression techniques tend to improve the estimation cost of the applications. However, the Ad-hoc technique remains closer to the real curve.

6.3. Results

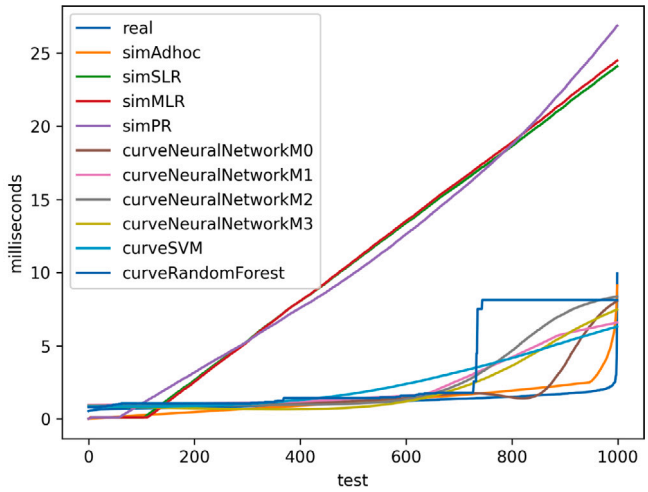
In this section we test our methodology with two real natural disasters occurred in Chile. The first one is a fire in Santa Olga in January of 2017. The second one is a tornado in Los Angeles (Bio Bio region in Chile) in May of 2019. We collected 3.281 tweets concerning the fire in Santa Olga between January 24 and January 28, 2017, and obtained 23.278 tweets related to the tornado in Los Angeles between May 29 and June 2, 2019. These tweets served as the basis for generating traces that emulate user behavior and subsequently execute tasks within the Ayni and Rimay applications. The creation of these traces was guided by insights from experts in evacuation processes.

Table 5 shows the results obtained with the tornado using the Rimay application. The first column is the execution order of the flows. The second column is the number of requests for the flow. Columns 3 to 7 show the results of the average times obtained with the simulator using different estimation techniques and results obtained with the real application deployed on a smaller-scale platform.

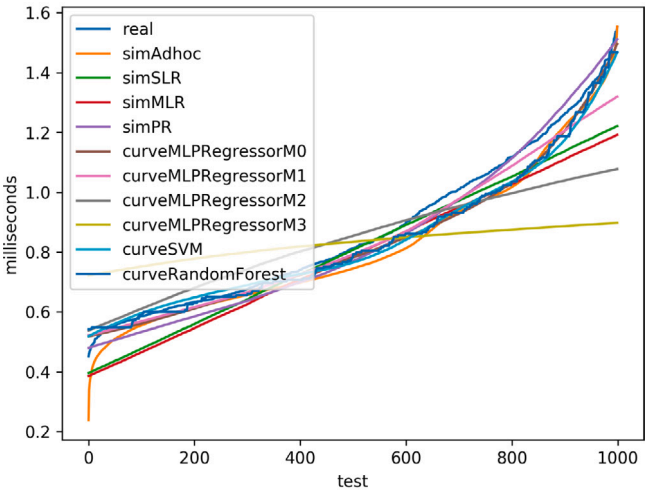
Similar to the previous case, Table 6 shows the results obtained with the tornado using the Ayni application for the fire hazard in 2017. The first column is the execution order of the flows. The second column is the number of requests for the flow. Columns 3 to 7 show the results of the average times obtained with the simulator using different estimation techniques and results obtained with the Ayni application deployed on a smaller-scale platform.

7. Discussion

The effective coordination of spontaneous volunteers engaged in rescue and assistance efforts during natural disasters requires organizations to rely on technology and software applications. These applications facilitate the analysis of rescue operation progress,



(a) LIST-MONGODB Work.



(b) CREATE-MONGODB Work.

Fig. 20. Execution time in milliseconds obtained with linear regression techniques, the Ad-hoc technique and no-linear regression techniques like the NN, SVM and Random Forest.

Table 6
Results obtained with Ayni for the fire of 2017 and results estimated with our simulator.

Flow Ayni	Tests	Time(ms)				
		Real	SLR	MLR	PR	Ad-hoc
A02	478	11,83	12,94	13,52	13,19	12,01
A01	239	45,62	34,70	34,30	34,71	45,27
A06	4064	0,95	1,003	0,97	1,05	1,01
A08	2980	22,26	21,74	21,50	22,08	21,86
A09	2235	83,78	74,33	73,67	75,10	81,69
A10	1490	56,54	64,38	63,54	64,68	55,81
A11	2953	13,005	14,28	14,48	15,20	13,43
A05	1105	0,93	1,01	0,98	1,04	0,97
A04	7734	0,94	0,99	0,96	1,03	0,97

critical states, resource availability, and more. Typically, such applications are designed considering the participating organizations, cultural context, and demographics of the areas in which they will be deployed. Given the high-stress situations people face and the imperative for fast responses, these applications must promptly fulfill user requirements, even in the event of potential failures in network and computational resources. This study specifically focuses on applications suitable for deployment on commodity hardware connected through the REUNA network infrastructure in Chile.[13]

The methodology presented in this work enables the prediction of software application performance designed for emergency scenarios during natural disasters. This approach involves modeling and simulating various components deployed on a distributed computing platform, leveraging machine learning techniques.

By comparing the predicted performance with the actual performance of applications deployed in the distributed infrastructure, we show that the proposed methodology accurately predicts application performance with minimal errors. Specifically, the Pearson correlation analysis reveals that the results for tasks executed by the Ayni and Rimay applications obtain correlations exceeding 90% in all cases. Therefore, our methodology effectively captures the trends of metrics observed in the real system. Additionally, results regarding the Root Mean Square Error (RMSE) and processing time in milliseconds show that our proposal replicates the performance reported by the real applications with an average error rate of 12%, considering results across all tasks in both applications. The average standard deviation of this error is 4,25. It is important to notice that the simulator offers the flexibility to easily switch between different prediction techniques. Therefore, it can include new prediction approaches without modifying the proposed methodology. The new approaches are implemented within the Work sub-class of the simulator.

To properly apply our proposed methodology, it is important to acknowledge that:

- Whenever new components, whether they be applications or computational resources, are integrated into the system, it is imperative to thoroughly characterize them through a meticulous process of analysis and testing. This involves the execution of benchmarks to assess their performance, stability, and compatibility with the existing infrastructure. This characterization process can be undertaken during periods of peacetime, in other words, when the system is not actively engaged in responding to a natural disaster. By conducting these assessments during non-crisis periods, organizations can ensure that the introduced components seamlessly integrate into the existing framework, minimizing potential disruptions during actual emergency situations. This proactive approach also allows for the identification and mitigation of any issues or bottlenecks, contributing to the overall resilience and efficiency of the system when it is most needed.
- Executing benchmarks requires a carefully controlled environment where the components of the applications under analysis are isolated to prevent interference from unrelated processes. This isolation ensures that the benchmark results accurately reflect the performance of the specific components being assessed, free from external factors that could introduce variability or skew the outcomes. In this controlled setting, the benchmarks can be run systematically, allowing for a comprehensive evaluation of efficiency, response times, and resource utilization. Nevertheless, the model may overlook the potential influence of other concurrently running applications sharing the same resources, particularly when these external applications compete for identical physical resources.
- For prediction techniques based on machine learning, it is crucial to manage overfitting. This is necessary to ensure the effectiveness of such techniques in predicting and generalizing to new data, thereby avoiding the capture of spurious patterns present in the training data. To tackle this problem it is possible during non-crisis periods to increase the size of the dataset used to train the techniques. On the other hand, in the case of using the ad-hoc technique, whenever an application, component, or resource is introduced, the engineer overseeing the system must verify that the statistical distributions remain valid.

8. Conclusions

We presented a methodology to estimate software application performance. The applications are designed to be used by administrators and volunteers in emergencies situations resulting from natural disasters like a tsunami, a tornado, or a fire. The infrastructure used to deploy the software applications leverages container technology and container orchestration, enabling workload distribution across multiple computing servers, dynamic scalability to handle increases in service demand, fault tolerance and resilient.

Our methodology involves five steps which helps to organize the prediction process and ensure effective deployment on a distributed platform. (1) It requires to identify the specific characteristics (scalability, fault tolerance, and resource utilization) that applications must satisfy to be successfully deployed on the distributed platform. (2) It also requires to run benchmarks, (3) define the available hardware resources within the distributed platform, (4) map out how tasks are executed across different components of the applications. This includes understanding the dependencies between tasks, parallel processing capabilities and resource allocation strategies. (5) We finally simulate the applications running on the distributed platform to predict their performance. The simulation considers different workload scenarios, potential failures, and resource utilization patterns. These steps allow to obtain a comprehensive understanding of the applications and the distributed platform. Additionally, our methodology incorporates a specification language used to formalize the modeling of application components, services, and resources. This language ensures clear and precise representation of the applications.

The simulation can be used with different techniques for estimating resource usage costs. In this work we use machine learning techniques and a statistic ad-hoc technique. However, other techniques can be easily implemented.

[13] https://www.reuna.cl/infraestructura-digital/#red-nacional

We conducted an evaluation of our proposal using two real applications, Ayni and Rimay. The results indicate that the ad-hoc estimation technique achieved the lowest error rates, ranging from 2.2% to 7.7% for Ayni and 1.7% to 5.5% for Rimay. In contrast, other techniques reported an average error between 12.96% and 15.89%. Additionally, there is a strong correlation between the predicted performance and the actual performance reported by the applications. Therefore, our study shows that our proposal can accurately estimate the performance of complex applications running on distributed platforms in critical emergency scenarios.

As future work, we plan to apply the proposed methodology to stream processing applications, which are useful to make online summaries of information as the data arrives. We also plan to use the proposed methodology for capacity planning in large data centers, which consists of searching for the deployment of applications that best take advantage of the resources available from the computing infrastructure and network.

Data availability

Data will be made available on request.

Acknowledgment

This work has been partially funded by the Agencia Nacional de Investigación y Desarrollo de Chile (ANID) under grant Basal Centre CeBiB code FB0001.

References

[1] J. MeenaManu, V. Vardhan, Efficient utilization of commodity computers in academic institutes: A cloud computing approach, in: XIII International Conference on Computer Science, Information Systems and Communication Technologies, Vol. 9, 2015, pp. 498–503.
[2] D. Regassa, H. Yeom, Y. Son, Harvesting the aggregate computing power of commodity computers for supercomputing applications, Appl. Sci. 12 (10) (2022).
[3] Cloud computing in natural hazard modeling systems: Current research trends and future directions, Int. J. Disaster Risk Reduct. 38 (2019) 101188, http://dx.doi.org/10.1016/j.ijdrr.2019.101188.
[4] X. Andrade, F. Layedra, C. Vaca, E. Cruz, RiSC: Quantifying change after natural disasters to estimate infrastructure damage with mobile phone data, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 3383–3391.
[5] H. Maryam, M.A. Shah, Q. Javaid, M. Kamran, A survey on smartphones systems for emergency management (SPSEM), Int. J. Adv. Comput. Sci. Appl. 7 (6) (2016) 301–311.
[6] X. Song, Q. Zhang, Y. Sekimoto, R. Shibasaki, Prediction of human emergency behavior and their mobility following large-scale disaster, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 5–14.
[7] E. Rosas, N. Hidalgo, V. Gil-Costa, C. Bonacic, M. Marin, H. Senger, L. Arantes, C. Marcondes, O. Marin, Survey on simulation for mobile ad-hoc communication for disaster scenarios, J. Comput. Sci. Tech. 31 (2) (2016) 326–349.
[8] M. Dorasamy, M. Raman, S. Muthaiyah, M. Kaliannan, Knowledge management systems for emergency managers: Malaysian perspective, in: 2011 International Conference on Semantic Technology and Information Retrieval, IEEE, 2011, pp. 289–296.
[9] K. Kremer, Anticipative interfaces for emergency situations, Inf. Des. J. 23 (1) (2017) 32–38.
[10] M. Liposinovic, Usability principles for (Re) design of user interface of emergency handling programs: Case study on a tool for decision support amidst a nuclear emergency: RASTEP, 2020.
[11] D. Velev, P. Zlateva, Principles of cloud computing application in emergency management, 25, IPEDR, IACSIT Press, Singapore, 2011, pp. 119–123.
[12] K. Ujjwal, S. Garg, J. Hilton, J. Aryal, N. Forbes-Smith, Cloud computing in natural hazard modeling systems: Current research trends and future directions, Int. J. Disaster Risk Reduct. 38 (2019) 101188.
[13] G. Ovando-Leon, L. Veas-Castillo, V. Gil-Costa, M. Marin, Bot-based emergency software applications for natural disaster situations, Future Internet 14 (3) (2022) http://dx.doi.org/10.3390/fi14030081.
[14] A. Matsunaga, J.A. Fortes, On the use of machine learning to predict the time and resources consumed by applications, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2010, pp. 495–504.
[15] T.M. Mitchell, Machine Learning, Vol. 45, (37) McGraw Hill, Burr Ridge, IL, 1997, pp. 52–78.
[16] T.M. Mitchell, Machine Learning, Vol. 45, (37) McGraw Hill, Burr Ridge, IL, 1997, pp. 230–247.
[17] M. Amarís, R.Y. de Camargo, M. Dyab, A. Goldman, D. Trystram, A comparison of GPU execution time prediction using machine learning and analytical modeling, in: Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on, IEEE, 2016, pp. 326–333.
[18] T.-P. Pham, J.J. Durillo, T. Fahringer, Predicting workflow task execution time in the cloud using a two-stage machine learning approach, IEEE Trans. Cloud Comput. 8 (1) (2020) 256–268, http://dx.doi.org/10.1109/TCC.2017.2732344.
[19] G. Yeung, D. Borowiec, A. Friday, R. Harper, P. Garraghan, Towards {GPU} utilization prediction for cloud deep learning, in: 12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20), 2020.
[20] M. Amiri, L. Mohammad-Khanli, Survey on prediction models of applications for resources provisioning in cloud, J. Netw. Comput. Appl. (2017).
[21] N. Balasubramanian, G. Kumaran, V.R. Carvalho, Predicting query performance on the web, in: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2010, pp. 785–786.
[22] Y. Zhang, J. Guo, E. Blais, K. Czarnecki, Performance prediction of configurable software systems by fourier learning (T), in: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, 2015, pp. 365–373.
[23] O. Rojas, V. Gil-Costa, M. Marin, Running time prediction for web search queries, in: Parallel Processing and Applied Mathematics, Springer, 2016, pp. 210–220.
[24] O. Rojas, V. Gil-Costa, M. Marin, A DFT-based running time prediction algorithm for web queries, Future Internet 13 (8) (2021) 204.
[25] G.A. Wainer, Discrete-Event Modeling and Simulation: A Practitioner's Approach, CRC Press, 2017.
[26] V. Gil-Costa, J. Lobos, R. Solar, M. Marin, Ameds-tool: an automatic tool to model and simulate large scale systems, in: Proceedings of the 2014 Summer Simulation Multiconference, 2014, pp. 1–8.
[27] S. Andreyev, User-driven applications, 2010, arXiv preprint arXiv:1004.0438.
[28] I.H. Sarker, K. Salah, Appspred: predicting context-aware smartphone apps using random forest learning, Int. Things 8 (2019) 100106.
[29] S. Zhao, Z. Luo, Z. Jiang, H. Wang, F. Xu, S. Li, J. Yin, G. Pan, AppUsage2Vec: Modeling smartphone app usage for prediction, in: 2019 IEEE 35th International Conference on Data Engineering, ICDE, IEEE, 2019, pp. 1322–1333.

[30] C. Bonacic, M. Marin, Simulation study of multi-threading in web search engine processors, in: International Symposium on String Processing and Information Retrieval, Springer, 2013, pp. 37–48.
[31] J. Orellana, C. Bonacic, M. Marín, V. Gil-Costa, Energysim: an energy consumption simulator for web search engine processors, in: Proceedings of the Summer Simulation Multi-Conference, Society for Computer Simulation International, 2017, p. 18.
[32] G. Ovando-Leon, L. Veas-Castillo, M. Marin, V. Gil-Costa, A simulation tool for a large-scale nosql database, in: 2019 Spring Simulation Conference (SpringSim), IEEE, 2019, pp. 1–12.
[33] F. Bai, A. Helmy, A survey of mobility models, in: Wireless Adhoc Networks, 206, University of Southern California, USA, 2004, p. 147.
[34] S.C. Nelson, A.F. Harris III, R. Kravets, Event-driven, role-based mobility in disaster recovery networks, in: Proceedings of the Second ACM Workshop on Challenged Networks, ACM, 2007, pp. 27–34.
[35] N. Aschenbruck, E. Gerhards-Padilla, M. Gerharz, M. Frank, P. Martini, Modelling mobility in disaster area scenarios, in: Proceedings of the 10th ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, ACM, 2007, pp. 4–12.
[36] D. Costantini, M. Münch, A. Leonardi, V. Rocha, P.S. Mogre, R. Steinmetz, Role-based urban post-disaster mobility model for search and rescue operations, in: Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on, IEEE, 2012, pp. 900–907.
[37] L. Conceição, M. Curado, Modelling mobility based on human behaviour in disaster areas, in: International Conference on Wired/Wireless Internet Communication, Springer, 2013, pp. 56–69.
[38] D.B. Koch, P.W. Payne, A simple evacuation modeling and simulation tool for first responders, in: SoutheastCon 2015, IEEE, 2015, pp. 1–5.
[39] H. Barbosa, M. Barthelemy, G. Ghoshal, C.R. James, M. Lenormand, T. Louail, R. Menezes, J.J. Ramasco, F. Simini, M. Tomasini, Human mobility: Models and applications, Phys. Rep. 734 (2018) 1–74.
[40] T. Yabe, Y. Sekimoto, K. Tsubouchi, S. Ikemoto, Cross-comparative analysis of evacuation behavior after earthquakes using mobile phone data, PLoS one 14 (2) (2019) e0211375.
[41] G. Solmaz, D. Turgut, A survey of human mobility models, IEEE Access 7 (2019) 125711–125731.
[42] E. Toch, B. Lerner, E. Ben-Zion, I. Ben-Gal, Analyzing large-scale human mobility data: a survey of machine learning methods and applications, Knowl. Inf. Syst. 58 (3) (2019) 501–523.
[43] T. Yabe, N.K. Jones, N. Lozano-Gracia, M.F. Khan, S.V. Ukkusuri, S. Fraiberger, A. Montfort, Location data reveals disproportionate disaster impact amongst the poor: A case study of the 2017 puebla earthquake using mobilkit, 2021, arXiv preprint arXiv:2107.13590.
[44] F. Liu, P. Shu, J.C. Lui, AppATP: An energy conserving adaptive mobile-cloud transmission protocol, IEEE Trans. Comput. 64 (11) (2015) 3051–3063.
[45] Z. Jiang, S. Mao, Energy delay tradeoff in cloud offloading for multi-core mobile devices, IEEE Access 3 (2015) 2306–2316.
[46] H. Wu, Y. Sun, K. Wolter, Energy-efficient decision making for mobile cloud offloading, IEEE Trans. Cloud Comput. 8 (2) (2018) 570–584.
[47] S. Nanda, C.R. Panigrahi, B. Pati, Emergency management systems using mobile cloud computing: A survey, Int. J. Commun. Syst. (2020) e4619.
[48] P. Cong, J. Zhou, L. Li, K. Cao, T. Wei, K. Li, A survey of hierarchical energy optimization for mobile edge computing: A perspective from end devices to the cloud, ACM Comput. Surv. 53 (2) (2020) 1–44.
[49] J. Ko, Y.-J. Choi, R. Paul, Computation offloading technique for energy efficiency of smart devices, J. Cloud Comput. 10 (1) (2021) 1–14.
[50] O. Egwuche, M. Ganiyu, M. Ibiyomi, A survey of mobile edge computing in developing countries: challenges and prospects, J. Phys.: Conf. Ser. 2034 (1) (2021) 012004.
[51] M.M. Alqarni, A. Cherif, E. Alkayal, A survey of computational offloading in cloud/edge-based architectures: Strategies, optimization models and challenges, KSII Trans. Int. Inf. Syst. (TIIS) 15 (3) (2021) 952–973.
[52] N. Liu, A. Haider, X.-H. Sun, D. Jin, Fattreesim: Modeling large-scale fat-tree networks for hpc systems and data centers using parallel and discrete event simulation, in: Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, ACM, 2015, pp. 199–210.
[53] A. Varga, R. Hornig, An overview of the OMNeT++ simulation environment, in: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 60.
[54] G.F. Riley, T.R. Henderson, The ns-3 network simulator, Model. Tools Netw. Simul. (2010) 15–34.
[55] T. Goyal, A. Singh, A. Agrawal, Cloudsim: simulator for cloud computing infrastructure and modeling, Procedia Eng. 38 (2012) 3566–3572.
[56] G. Keller, M. Tighe, H. Lutfiyya, M. Bauer, DCSim: A data centre simulation tool, in: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, IEEE, 2013, pp. 1090–1091.
[57] S.K. Gupta, A. Banerjee, Z. Abbasi, G. Varsamopoulos, M. Jonas, J. Ferguson, R.R. Gilbert, T. Mukherjee, Gdcsim: A simulator for green data center design and analysis, ACM Trans. Model. Comput. Simul. (TOMACS) 24 (1) (2014) 3.
[58] A. Akram, L. Sawalha, A survey of computer architecture simulation techniques and tools, Ieee Access 7 (2019) 78120–78145.
[59] B. Burns, J. Beda, K. Hightower, Kubernetes, Dpunkt Heidelberg, Germany, 2018.
[60] F. Soppelsa, C. Kaewkasi, Native Docker Clustering with Swarm, Packt Publishing Ltd, 2016.
[61] V. Farcic, The DevOps 2.1 Toolkit: Docker Swarm, Packt Publishing Ltd, 2017.
[62] M. Marzolla, et al., Libcppsim: a simula-like, portable process-oriented simulation library in C++, in: Proc. of ESM, Vol. 4, 2004, pp. 222–227.
[63] J. Turnbull, The Docker Book: Containerization Is the New Virtualization, James Turnbull, 2014.
[64] J. Carlson, Redis in Action, Simon and Schuster, 2013.
[65] B. Erinle, Performance Testing with JMeter 3, Packt Publishing Ltd, 2017.
[66] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning, vol. 112, Springer, 2013.