future internet

MDPI

Article

Bot-Based Emergency Software Applications for Natural Disaster Situations

Gabriel Ovando-Leon [1,*], Luis Veas-Castillo [1], Veronica Gil-Costa [2] and Mauricio Marin [1,3]

[1] CITIAPS, Universidad de Santiago de Chile, Santiago 9170020, Chile; luis.veasc@usach.cl (L.V.-C.); mauricio.marin@usach.cl (M.M.)
[2] CONICET, Universidad Nacional de San Luis, San Luis 5700, Argentina; gvcosta@unsl.edu.ar
[3] CeBiB, Center for Biotechnology and Bioengineering, Santiago 9170020, Chile
* Correspondence: juan.ovando@usach.cl

Abstract: Upon a serious emergency situation such as a natural disaster, people quickly try to call their friends and family with the software they use every day. On the other hand, people also tend to participate as a volunteer for rescue purposes. It is unlikely and impractical for these people to download and learn to use an application specially designed for aid processes. In this work, we investigate the feasibility of including bots, which provide a mechanism to get inside the software that people use daily, to develop emergency software applications designed to be used by victims and volunteers during stressful situations. In such situations, it is necessary to achieve efficiency, scalability, fault tolerance, elasticity, and mobility between data centers. We evaluate three bot-based applications. The first one, named Jayma, sends information about affected people during the natural disaster to a network of contacts. The second bot-based application, Ayni, manages and assigns tasks to volunteers. The third bot-based application named Rimay registers volunteers and manages campaigns and emergency tasks. The applications are built using common practice for distributed software architecture design. Most of the components forming the architecture are from existing public domain software, and some components are even consumed as an external service as in the case of Telegram. Moreover, the applications are executed on commodity hardware usually available from universities. We evaluate the applications to detect critical tasks, bottlenecks, and the most critical resource. Results show that Ayni and Rimay tend to saturate the CPU faster than other resources. Meanwhile, the RAM memory tends to reach the highest utilization level in the Jayma application.

Keywords: natural hazards; applications for natural disasters; risk management; bots

Citation: Ovando-Leon, G.; Veas-Castillo, L.; Gil-Costa, V.; Marin, M. Bot-Based Emergency Software Applications for Natural Disaster Situations. *Future Internet* **2022**, *14*, 81. https://doi.org/10.3390/fi14030081

Academic Editor: Vijayakumar Varadarajan

Received: 29 December 2021
Accepted: 29 January 2022
Published: 9 March 2022

1. Introduction

During the last few years, there has been a special interest in the development of software applications focused on providing help during and after the occurrence of natural disasters of great magnitude [1–3]. The normal life of people is destabilized, there is chaos and the need for help, rescue, and relief to the victims of the catastrophe. All of this can cause human, structural, and economic losses that can rise incalculably if not managed properly. In this crisis context, emergency response teams composed of spontaneous volunteer and experts have a key role in helping reduce the impact of the natural disasters. They are typically coordinated by social, public, and national organizations responsible for sending the appropriate resources to the most affected areas and also for reporting the news. Emergency response teams need to work fast, and the decision makers must base their decision on the collected data from different sources as soon as possible [4].

Nowadays, technology such as mobile devices and smartphones have driven the development of applications devised for large-scale natural disasters to aid victims and also to facilitate the collaboration between spontaneous volunteers [5,6]. However, in a serious emergency situation, people will always turn to the software they use every day,

the software that is most familiar to them. It is unlikely that a person who can potentially participate as a spontaneous volunteer in the face of a serious emergency remembers in advance to install and learn how to use a large-scale disaster application. Especially in a stressful situation, either as an affected person or as a person in charge of volunteer work or being part of an emergency team, people will tend to turn to the software they know best and have available on their smartphones [7,8]. Social media has also been successfully used to rapidly evaluate the impacts of disasters in real time [9,10].

1.1. Research Objectives

In this paper, we analyze the feasibility of using bot-based software applications for emergencies devised to assist social organizations that are activated upon the occurrence of large-scale natural disasters. The applications are executed on commodity hardware usually available from public organizations, such as universities, during the emergency situation. 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.

This problem is challenging, since we need to understand how the application behaves under high rates of workloads, which components can became bottlenecks, and which are the saturation points of the applications to then efficiently deploy them to support unexpected peaks of user requests. We investigate how the communication affects the performance of each task of the applications, which resources (CPU, RAM, network) saturate faster, and how can we use replicas to support higher workload. We analyze the capability of bot-based software to achieve efficiency, scalability, fault tolerance, elasticity, and mobility in commodity data centers by using container technology.

1.2. Contribution

We present and evaluate three bot-based applications that can help organizations help citizens who suffered from natural disasters. Bots have been previously used for medical purposes [11], for communication and training [12], and also for crisis communication and management [13]. However, to the best of our knowledge, there is no previous work evaluating the performance of bot-based applications to show that those applications are able to cope with critical mission requirements. In particular, we focus on critical resources and critical tasks when stressing the systems to unexpected peaks of user requests in natural disasters situations.

The three bot-based applications are named Jayma, Ayni, and Rimay. Jayma gives supports to people who are in dangerous situations during a natural disaster. For example, it informs a group of predefined contacts whether the person is in danger or not and the location where he/she is. Ayni coordinates teams (small and large) of volunteers in the field. Ayni assigns tasks to volunteers and receives status reports and additional information. Rimay coordinates tasks assigned to organizations that provide support in a natural disaster situation. Users access these applications through bots of Telegram and Messenger. This is relevant because bots are a mechanism to integrate into the software that people use as part of their lives. These bot-based applications can be used by victims and also by volunteers to coordinate relief work.

We deploy the applications on a computational platform based on container technology, container orchestration, and virtual machines. Containers allow the deployment of applications independently of the operating system, since it encapsulates the applications and their dependencies in portable modules that run under the same operating system kernel. Containers facilitate the deployment of applications in a wide range of operating systems such as Linux, Windows, or Mac. This container orchestration technology guarantees application mobility and also ensures fault tolerance through data replication and volatile applications where data do not need to be recovered after a failure.

We present a comprehensive performance evaluation study on a single processor and on multiple processors with different workloads. For each application, we detect the most

critical tasks, meaning the tasks that saturate faster supporting fewer requests per unit of time. We also evaluate the utilization levels of the CPU, RAM, and communication network. This information can be used by the engineer in charge of the data center to efficiently deploy the applications. The results show that the CPU is the most critical resource in Ayni and Rimay reporting utilization levels close to 60–70%. Meanwhile, the Jayma application tends to saturate faster the RAM memory than other resources.

1.3. Outline

The remaining of this paper is organized as follows. Section 2 describes the computational platform and the technologies used to deploy the applications. Section 3 present the bot-based applications: Ayni, Rimay, and Jayma. Section 4 presents the experimental results. Section 5 presents related work and summarizes the features of our applications. Section 6 presents a discussion about the limitations of our work. Section 7 concludes and present future work.

2. Platform Design

The design of our platform includes four main components: (1) a Front-end, (2) a Back-end, (3) a Bot Back-end which is an API REST that responds to user requests from the Messenger chat of Facebook and/or Telegram, and (4) a Data Repository like MongoDB [14]. Each component can be replicated, distributed, and partitioned based on the workload and communication requirements. Users connect to the Front-end using a smartphone, a tablet, or a computer. The Back-end hosts the applications and communicates with the data repository component. The Back-end performs data searches and executes transactions. The Data Repository stores the data in non-relational databases such as MongoDB [14] or Cassandra [15]. Both the Back-end and the Data Repository can be deployed on different servers. Our platform uses a container [16] and container technology [17], which allow the deployment of applications based on microservices, to achieve auto-scalable, fault-tolerant, and stateless systems, making the configuration of the infrastructure transparent for the programmer. In particular, we use Kubernetes [18] and Docker [19] due to their wide dissemination in the technology community and the support available in the software communities. In Figure 1, we show the general scheme of the architecture, where Nginx [20] is the access point, which is the application server and works as a proxy. The container orchestrator manages and redirects requests to a container. Finally, the figure shows the interaction between different Back-ends and different Data Repositories such as MongoDB, Cassandra, MariaDB, etc. The orchestrator also starts new instances of the applications running in the containers. New instances are started when failures occur or when more resources are needed to support user demand. To this end, ImageRepo maintains an undeployed image of all orchestrated containers. As a particular example, Figure 2 shows the deployment of the complete platform distributed among a cluster of servers coordinated by a container orchestrator.

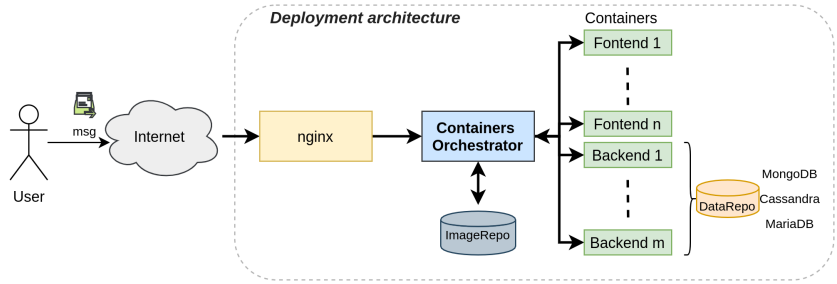


Figure 1. General scheme of the proposed architecture.

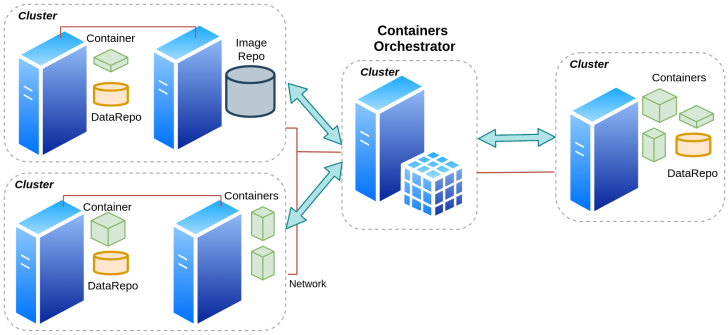


Figure 2. Example of a distributed platform.

Figure 3 summarizes the technologies used to implement the bot-based application on the platform. Our platform supports different types of configurations: unshared, virtual machine monitor (VMM), cloud computing, and container. In the non-sharing configuration, we install the operating system (OS) that manages the resources, the run-time libraries, and then, we deploy the applications. Another possibility is to connect the hardware directly to the VMM (hypervisor) independent of the operating system. The star ($*$) symbol shows this configuration in Figure 3. Cloud computing provides tools to manage virtual machines and their resources, which facilitates tasks such as self-scaling (elastic), live migration, and resizing a virtual machine based on workload. Similarly, container technology and orchestrator provide tools to create container-based, self-scaling (elastic) microservices. It automatically manages the containers using a load-balancing algorithm. Using containers without orchestrator requires manual intervention (on the host VM) to manage the containers.

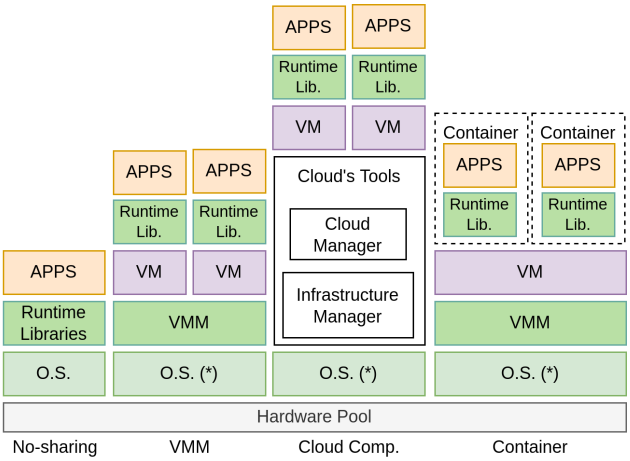


Figure 3. Virtualization technology used in our proposed platform.

3. Bot-Based Applications for Natural Disasters

In this section, we describe the design of the bot-based applications evaluated in this work: that is, the Jayma, Ayni, and Rimay applications.

3.1. Jayma Application

After a natural disaster strikes, Jayma (https://citiaps.usach.cl/portafolio/jayma, accessed on 28 January 2022) gives information about safe places and meeting points, recommends routes to follow, and spreads information through the Messenger bot. For the Messenger bot, the app creates a network of the user's contacts. These contacts are included in the Jayma network after they agree to participate by using the Jayma app.

The Back-end of the application supports a website for the administrator. The administrator can send disaster alerts to users. When the administrator sends an alert message,

users receive a notification (Facebook Messenger message) indicating that a natural disaster has occurred. The administrator can also submit relevant information such as safe points. Figure 4 shows some screenshots of the Jayma bot. The application supports the following tasks:

1. Send alerts: The administrator sends alert messages to the users included in the Jayma network.
2. Access to the main menu: Lists all the application options (Figure 4a).
3. Information: Provides information on meeting points or safe areas. These are safe geographic locations (latitude and longitude) and are activated by the administrator. Upon accessing the link, the bot delivers a Google Maps map with the last known location and the route to the closest meeting point (Figure 4b–d).
4. Report user status: The bot asks the user for the location. Once the bot receives the location, it asks the user if they are well. The bot notifies the user's contacts of their status and location. In case the user is unwell, the bot asks what is the problem (panic/injured/infrastructure problems/needs supplies). See Figure 4e,f.
5. Contact management: Adds new contacts to the Jayma network. It also allows deleting a contact and listing current contacts. To include a person as a new contact, the app sends a message to that person with a link to the bot.

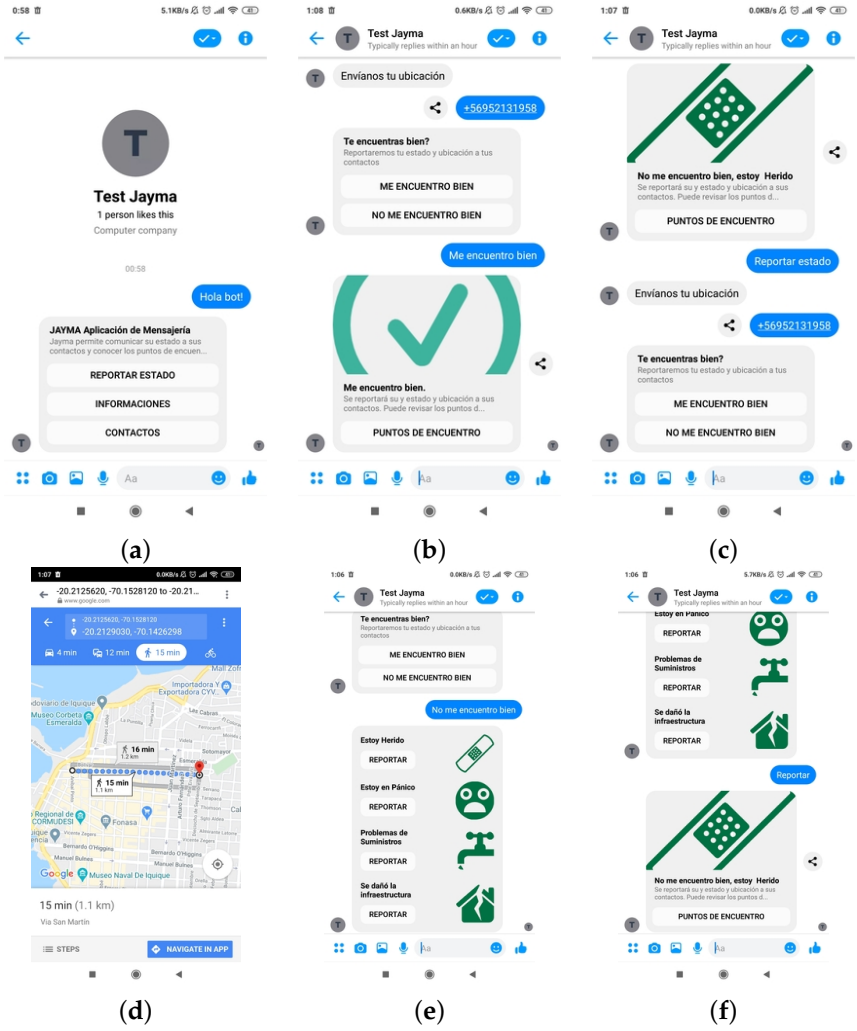


Figure 4. Screenshots of different tasks provided by the Jayma bot. (**a**) Main menu, (**b**) user status information, (**c**) meeting points, (**d**) evacuation route, (**e**,**f**) reports user problems.

3.2. Rimay Application

Rimay (https://citiaps.usach.cl/portafolio/rimay, accessed on 28 January 2022) 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. Figures 5 and 6 show some screenshots of the Rimay bot.

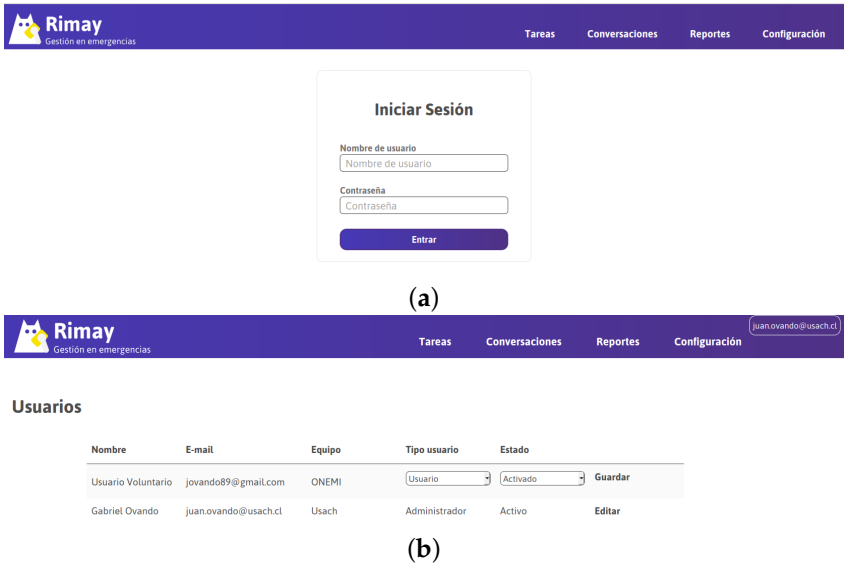


Figure 5. Screenshots of Rimay's Front-end application: (**a**) login web page, (**b**) information about users.

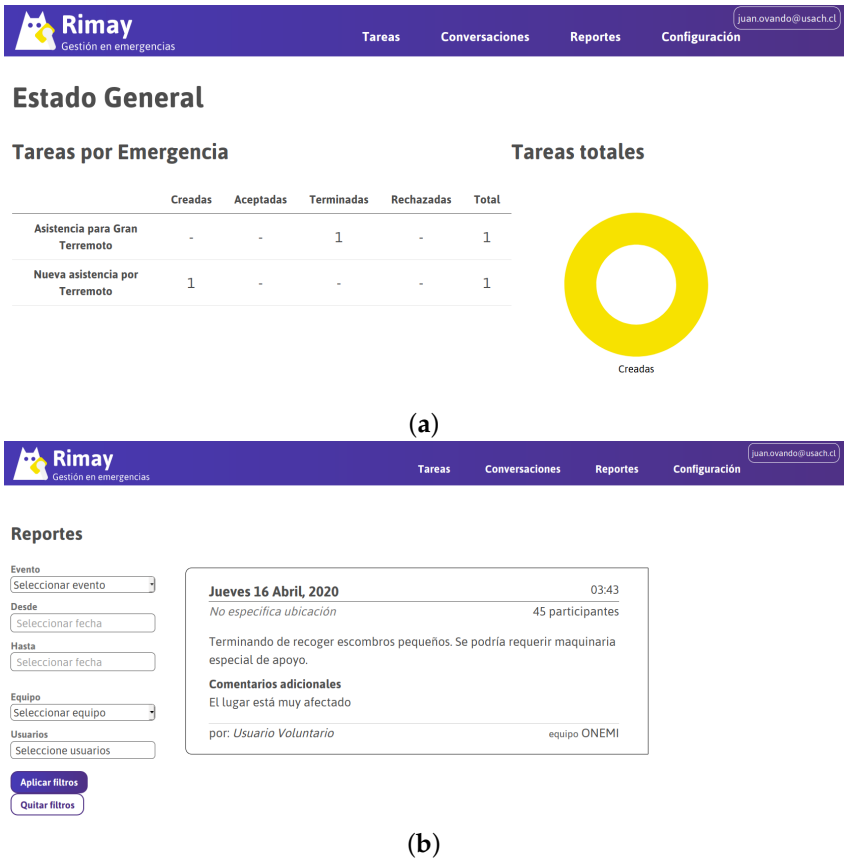


Figure 6. Screenshots of Rimay's Front-end application: (**a**) statistics report on events and (**b**) reports.

The application supports the following tasks:

1. Enrollment: Users request to register as volunteers. When the coordinator accepts the request, the data associated with the volunteer are stored in the system. In addition, the coordinator can assign the coordinator role to new users (Figure 5a,b).
2. Create event: The coordinator creates an event such as a forest fire. In addition, the coordinator can create tasks for each event, such as distributing face masks. Tasks have information such as where the face masks are stored and a job description. Volunteers can accept or decline assignments. Volunteers can also finish assignments when they are completed. The coordinator can also finish the event when it is complete. (Figure 6a).
3. Delete user: The coordinator can delete users from the system.
4. Send file: The coordinator can send a file to the volunteers.
5. Help/Status: Reports the status of volunteers.
6. Send report: Volunteers can send a report (attached file) to the coordinator. See Figure 6b.

Figure 7 shows the steps executed during the execution of different tasks on the Rimay application. In particular, in this example, the coordinator sends the command "newTask" to the Telegram bot to create a new task. The bot asks for the name of the task. After the coordinator answers, the bot asks if he/she wants to upload a file. The coordinator uploads an image file. Then, the bot asks if he/she want to add a GPS location and the coordinator responds by sharing a geolocation. The bot displays a menu showing the existing volunteers. Then, the coordinator selects a volunteer who will receive the new task. Finally, the coordinator sends a request to the Back-end component to verify the information and to store it in a database. The boxes S_i with $i = 1, 2, 3, 4, 5$ represent the time it takes for the person to respond. The boxes $Tbot_i$ with $i = 1, 2, 3, 4, 5$ represent the processing times of the bot. Tbe_1 represents the processing time of the Back-end. Notice that during the execution time Tbe_1, the bot thread is free until it gets the response. Figure 5 shows screenshots of the login web page and the information provided the Rimay application running on the Front-end.

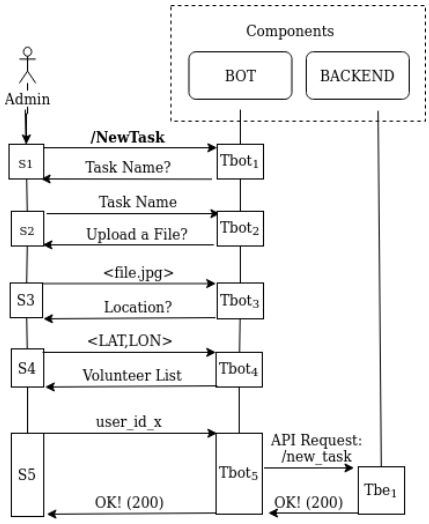


Figure 7. Steps executed by different tasks on the Rimay app. The coordinator and the volunteers talk through the Telegram bot.

3.3. Ayni Application

The Ayni application (https://citiaps.usach.cl/portafolio/ayni, accessed on 28 January 2022) manages groups of volunteers through a bot of Telegram. These volunteers sign up for the current emergency and they inform about their characteristics (physics, skills, knowledge, etc.). Volunteers can participate in tasks created for a particular emergency.

Figure 8 shows the general scheme of the application. We use a cache because the bot needs to be stateless for replication and also needs to store information of a given task as the execution of the task advances in different components of the platform. The application supports the following tasks:

1. Create emergency: The coordinator creates a specific emergency and selects volunteers to participate in it.
2. View emergency details: Displays the specifications of an emergency.
3. Create Task: The coordinator creates tasks assigned to the emergency. In turn, it chooses volunteers who are asked if they want to participate in the assigned task.
4. Create volunteer: People register into the system.
5. Task details: Displays the task specifications.
6. Volunteer details: Displays the volunteers characteristics.
7. Volunteers agree to participate: A volunteer agrees to participate in an emergency.
8. Tasks per emergency: Displays the tasks for each emergency.
9. Active tasks for emergencies: Only active tasks assigned to emergencies are displayed.
10. Tasks completed by emergency: Only tasks completed by emergencies are displayed.
11. Volunteer accepts task: A volunteer accepts the task.
12. Volunteer completes task: A volunteer informs that the assigned task has been completed.

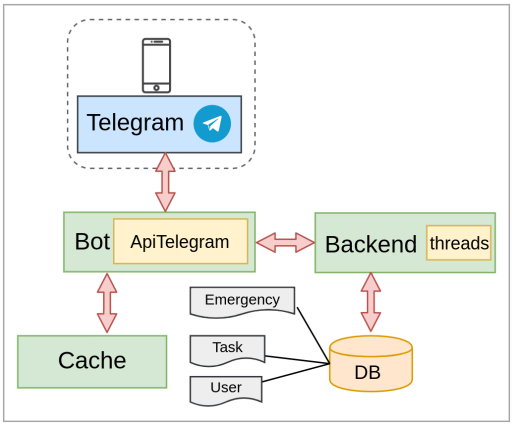


Figure 8. General scheme of the Ayni application.

4. Experiments

4.1. Platform Configuration

We evaluate the performance of the applications on a processor with 32 Cores, a RAM of 64 GB, HDD of 1.8TB, CPU MHz 1298. We deploy the applications on a single processor using Dockers. Figure 9 shows the configuration for a single processor. We use one VM to run the nginx, one for the back-end, and one virtual machine for the data repository. In particular, we use MongoDB as the data repository component. Each virtual machine has two cores, 4 GB of RAM, and 100 GB of hard disk. To detect bottlenecks and measure resource utilization, we use Jmeter 5.3 [21], which is a free software implemented in java.

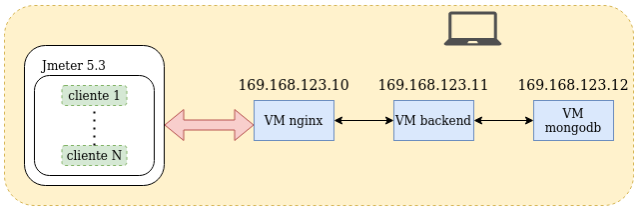


Figure 9. Deployment of the applications on a single processor.

We also evaluate the performance of the application on two processors with the same characteristics as described above. The bandwidth between both processors reported by

iperf (https://iperf.fr/, accessed on 28 January 2022) is 17 Gbits/s. Figure 10 shows the deployment of the apps in one of those processors. The Bot-back-end and the Back-end are worker components. Each worker has a virtual machine with 3 cores and 4 GB of RAM. Other components are executed on VM with two cores and 4 GB of RAM. The Master component executes Kubernetes. Only for the Ayni app, we use REDIS as a cache service to store the current state of the flow of each process. That is, if the task has a single action (such as help), only the first state of the task is saved and then deleted at the end of the flow of the task. However, if the task is longer, the flow of sub-tasks is saved until the complete flow is finished. The repoimage component is used by Kubernetes (it is the container image repository) when the deployment of the first container is performed. The applications are replicated on three workers, and to support fault tolerance when a worker fails, Kubernetes uses RepoImages to look for the corresponding image and deploy it in the selected worker. The validation of the tasks executed by the applications was based on expert consensus.

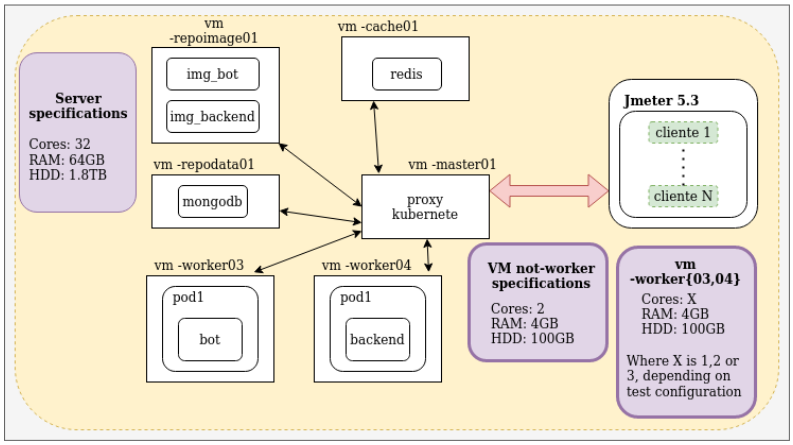


Figure 10. Deployment of the applications on two processors.

4.2. Bottlenecks and Performance Evaluation for Ayni

In this section, we evaluate the saturation points reported by different tasks executed in Ayni. To stress the app, we do not include the communication via the Telegram api because it limits the number of requests per second that we can send to the application to 30. In the following experiments, we send 600, 900, 1200, and 1500 requests to the application every 30 s. We deploy the application on a single processor according to Figure 9. We also show results of the Ayni application running on two processors according to Figure 10.

In Table 1, we show the tasks executed by the Ayni application and the number of requests per second supported by each task before it fails. We show that there are two critical tasks: "03—Create volunteers" and "06—Volunteer details". Both tasks support a maximum of 60 requests per second on a single processor and 100 requests when the Ayni application is deployed on two processors. With more requests, the tasks became unstable, and some requests are not completed. Other tasks support a larger number of requests, as in the case of "05—Task details", "07—Volunteers agree to participate" and "13—Volunteers accept tasks".

Figure 11 shows the requests processed successfully in a single processor (red line) and requests not completed (green line). We show the results obtained by the task "10—Tasks Completed by Emergency". The x-axis shows the advance of time in seconds. We divide the x-axis into segments of 30 s each. We start with 600 requests in the first time segment. Then, in the following segments, we increase by 300 the total number of requests. The results show that not all requests are completed in the fourth segment. That is, with an incoming rate of 40 requests per second (1200 request in 30 s), the system begins to saturate, and some requests are discarded. The system reports that 11.78% of the requests are not processed.

Table 1. Maximum number of requests supported by each task of the Ayni application on a single processor and with a distributed configuration.

Task	#Requests (Single)	#Requests (Distributed)
01 Create Emergency	100	180
02 Create Task	200	360
03 Create Volunteer	60	100
04 View Emergency Details	400	770
05 Task Details	400	760
06 Volunteer Details	60	100
07 Volunteers Agree to Participate	400	750
08 Tasks per Emergency	400	750
09 Active Tasks for Emergencies	400	750
10 Tasks Completed by Emergency	200	380
11 Volunteer Accepts Tasks	400	770
12 Volunteer Completes Tasks	200	380

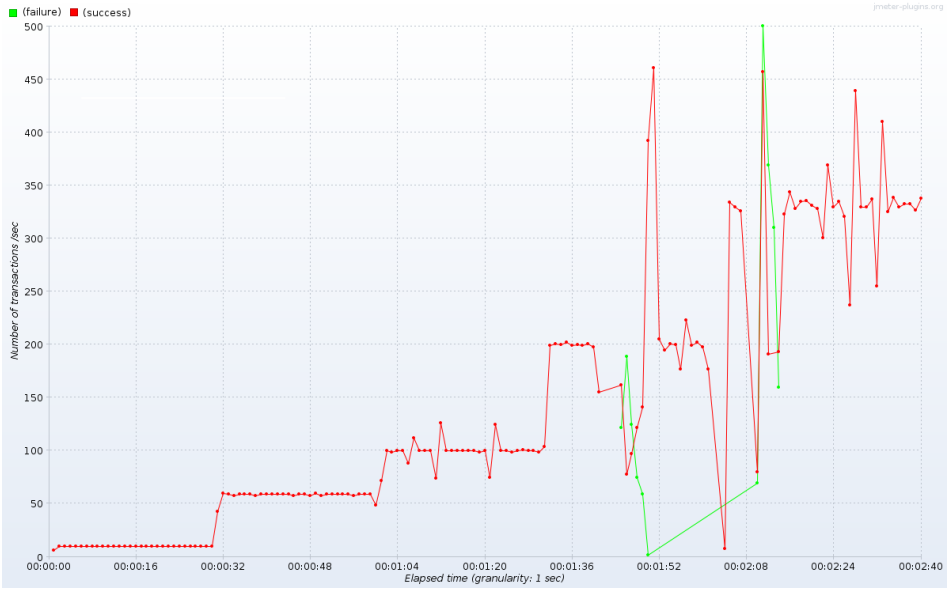
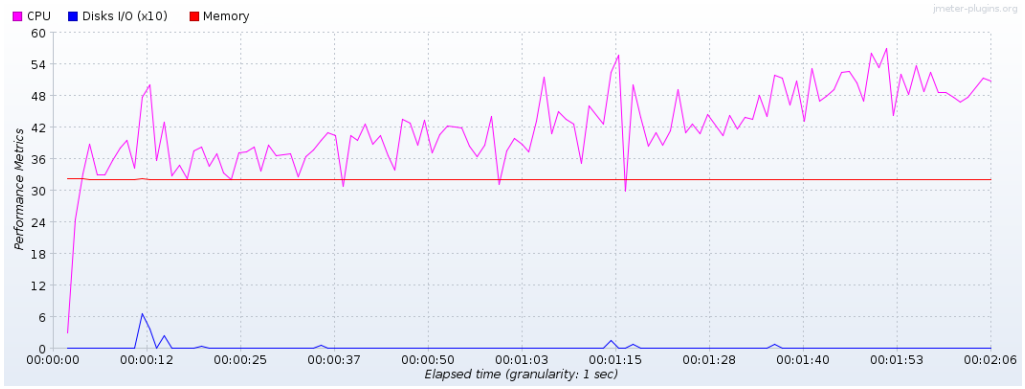


Figure 11. Single processor: Results obtained by the task "10—Tasks Completed by Emergency" on a single processor. The red line represents requests successfully completed. The green line represents requests not completed.

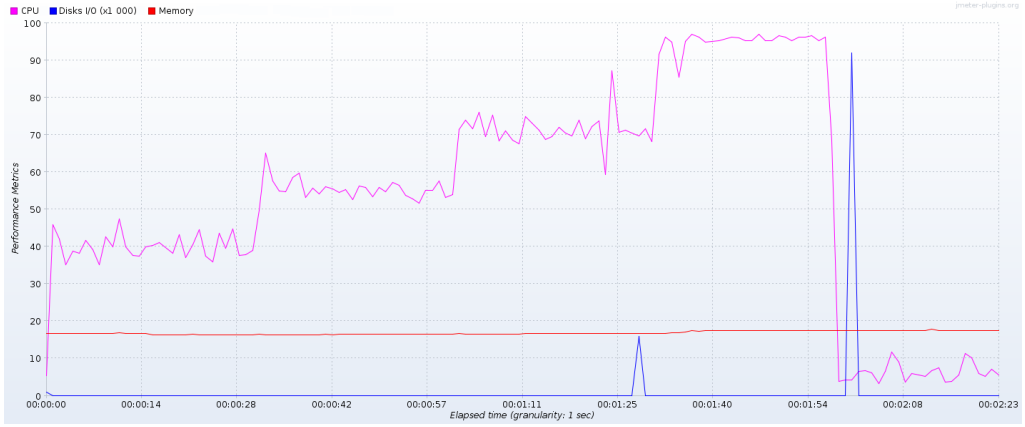
Figure 12 shows results obtained with a distributed deployment. In particular, we plot the utilization of different resources used by MongoDB (top) and the workers (bottom). Figure 12a shows that the CPU utilization reported by MongoDB reached peaks close to 60% in the last time interval. Similarly, Figure 12b shows the resource utilization reported by the workers. In this case, the CPU tends to be more demanding, reaching almost 100% of utilization in the last interval of time. On the other hand, the disk and memory utilization remains almost constant in both components (MongoDB and worker). Therefore, the CPU is the most critical resource for the Ayni application. We obtain a similar behavior with other tasks.

Figure 13 shows the communication cost reported by each of the components deployed to run Ayni in a distributed manner. The x-axis shows the time advance and the y-axis shows the communication cost in bytes. The blue line represents the component worker (an Ayni instance), which reports the highest load on the network, followed by the Master (brown line) and in third place is MongoDB (light blue line). These results show how the communication rate tends to increase every 30 s. This is because every 30 s, the number of requests entered into the system increases. Notice that the peaks in red color and dark blue

color are scaled ×100 for a better representation. Therefore, these peaks do not represent a high communication cost.



(**a**)



(**b**)

Figure 12. Distributed deployment: Resource utilization reported by (**a**) MongoDB and (**b**) the workers for the task: "10—Tasks Completed by Emergency".

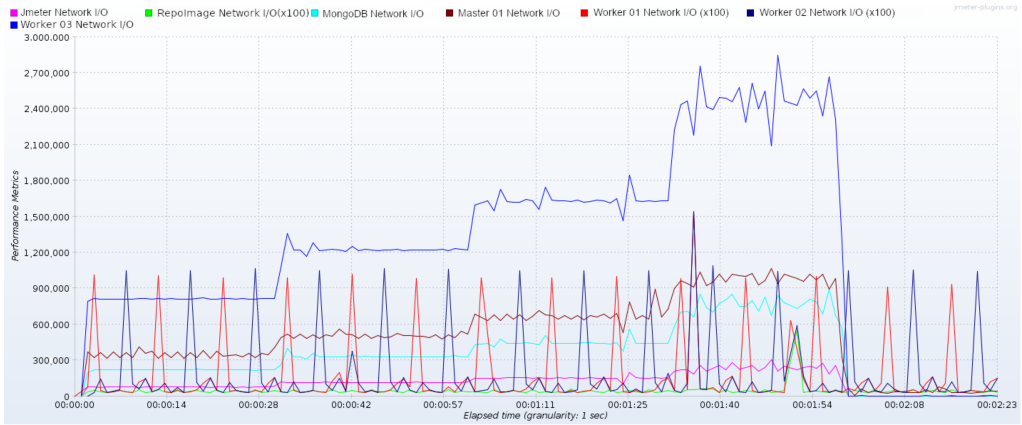


Figure 13. Network costs reported by the Ayni application deployed on two processors.

4.3. Bottlenecks and Performance Evaluation for Rimay

In this section, we present the performance evaluation of the Rimay application as the number of users increases every 30 s. Each user is a thread in the application and consecutively executes 10 requests. We set an interval of time of 3 s between requests. Different users can send requests in parallel.

In Table 2, we show results for an experiment with four time intervals of 30 s each. In the first time interval, we process requests of 100 users; in the second time interval, we

process requests of 200 users, and in the last time interval, we process requests of 600 users. In the second column of the table, we show the average response time in milliseconds (ms.) reported by each task. The average response time is computed as the average of the results obtained in each time interval. We also show the number of tasks processed per second (s.) and the execution time in seconds required to process all the requests injected in each time interval. As expected, as the number of simultaneous users increases, the response time increases due to requests accumulated.

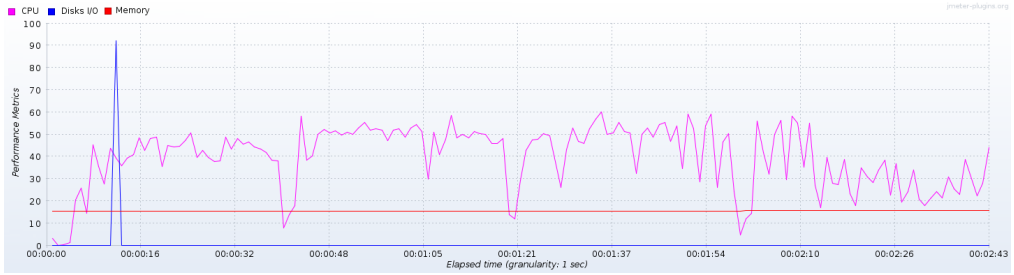
Table 2. Single processor: Execution time reported by different tasks executed by the Rimay application.

Task	Avg. Time (ms.)	# Tasks per s.	Exec. Time (s.)
Enrollment	88.93	2.72	132.28
		2.94	204.33
		3.03	276.49
		3.09	348.68
Create event	7.38	1.89	94.99
		2.1	143.2
		2.2	191.24
		2.25	239.28
Send report	112.03	3.85	109.03
		4.23	165.29
		4.42	221.4
		4.52	278.4
Help/Status	7.32	1.63	36.9
		2.22	45.05
		2.64	53.07
		2.95	61.09

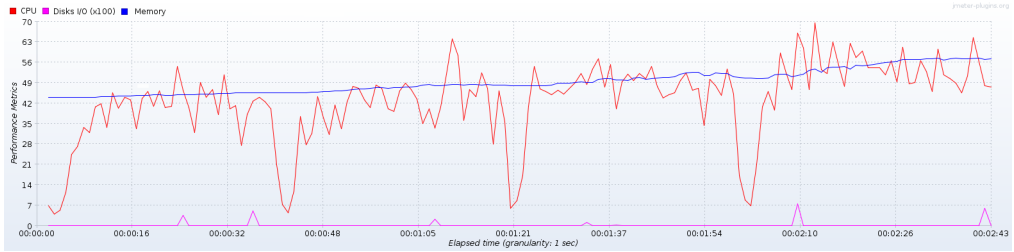
We also deploy the Rimay application on two processors as described in Figure 10. We use three cores for the workers and two cores for other components.

Figure 14 shows Rimay's performance when running the "Enrollment" task with a distributed deployment. The x-axis shows the time advance in seconds. The y-axis shows the utilization of resources (CPU, memory, and disk). During the first 30 s, the application processes 100 requests from users. In the next 30 s, the system processes 300 requirements. In the last time interval, from $x = 60$ s, the system processes 600 requests. Figure 14a shows that the resource utilization reported by MongoDB is below 50%. On the other hand, in Figure 14b, we show that the CPU and memory utilization in the workers are similar. The memory utilization slightly increases in each time interval, and the CPU reports peaks close to 70%.

Figure 14b presents down peaks, which are due to the execution of a new group of threads. For example, at the beginning of the second interval of 30 s, we change the configuration of the application from a group of 100 threads to a group of 300 threads. All the threads are executed in a time window of 30 s to avoid saturating the system with additional threads. This parameter is called the ramp-up period. Therefore, when a thread pool is started, T threads are running per second. In other words, in the first time interval, we use $T = 100$ threads to process the incoming requests. In the second time interval, we use $T = 300$ threads, and so on. Notice that in the second time interval, when the application processes 10 requests per second (300/30 s), the CPU become a critical resource. Therefore, this figure shows that the CPU tends to saturate in the Rimay application, reaching utilization levels of 90% of utilization. Others tasks report similar behavior.



(**a**)



(**b**)

Figure 14. Distributed deployment: Resource utilization reported by (**a**) MongoDB and (**b**) to workers for the task "Enrollment".

Figure 15 shows the number of bytes transferred through the network by each one of the components deployed to execute Rimay in a distributed manner. These results show how the communication rate tends to increase in stages every 30 s. This is because every 30 s, the number of requests increases. The Master component reports the highest communication costs because it has to periodically check for the status of other components. Note that the pink line is scaled $\times 1000$ for a better illustration of the graph.

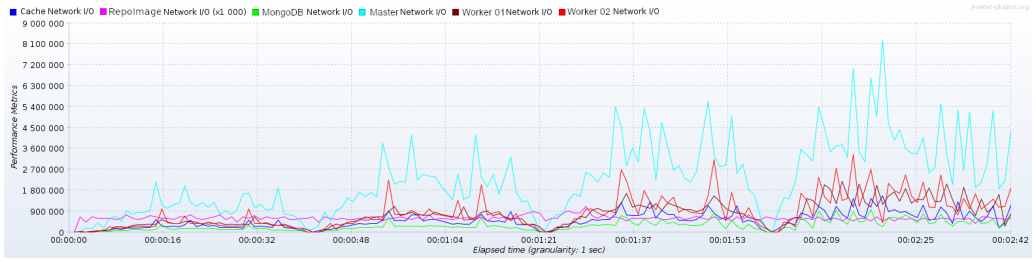


Figure 15. Communication cost in bytes reported by Rimay.

4.4. Bottlenecks and Performance Evaluation for Jayma

In this section, we show the performance of the Jayma application as we increase the number of users and therefore the number of simultaneous requests. In the distributed case, we deploy the workers that actually execute Jayma on virtual machines with three cores. MongoDB and the Master that executes components such as nginx use virtual machines with two cores.

In Table 3, we show the number of requests per second supported by each task before it fails. We show that the task "Contact management" supports a maximum of 120 requests per second on a single processor. However, when the application is deployed on two processors, the task supports up to 1100 requests. With more requests, the tasks become unstable, and some requests are not completed. Other tasks support a larger number of requests.

Figure 16 shows the utilization achieved by the CPU and the RAM memory for the task "Send alert" when the application is deployed on a single processor. At the beginning of the experiment, we process 600 requests. Then, for each interval of 30 s, we increase the number of requests to 900, 1200, and 1500 requests. The graph shows that the CPU remains below 50% of utilization. Only at the end of the experiment when the number of

requests tends to saturate the system, the CPU reaches utilization levels close to 100%. On the other hand, the utilization of the RAM memory tends to increase faster, reaching also 100% during the last interval of time.

Table 3. Number of requests supported by each task of Jayma on a single processor and in a distributed manner with two processors.

Task	#Requests (Single)	#Requests (Distributed)
Send alert	<1200	<1680
Access to the main menu	>1500	>1500
Information	>1500	>1500
Report user status	>1500	>1500
Contact management	<120	<1100

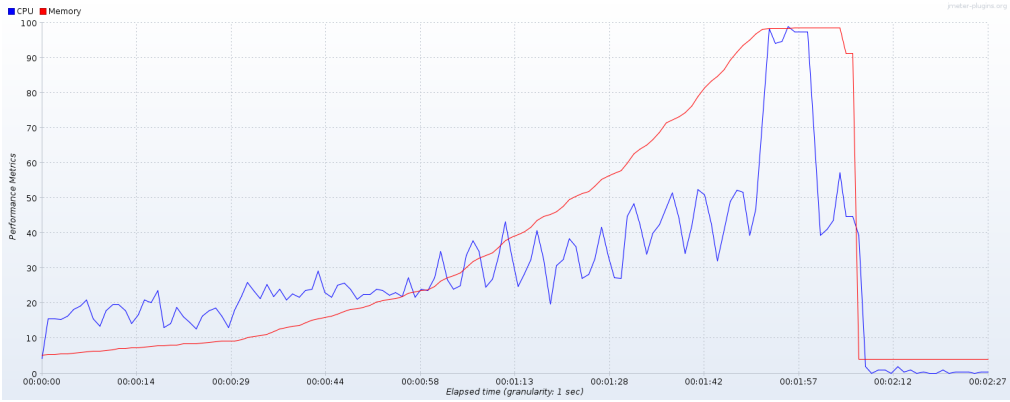
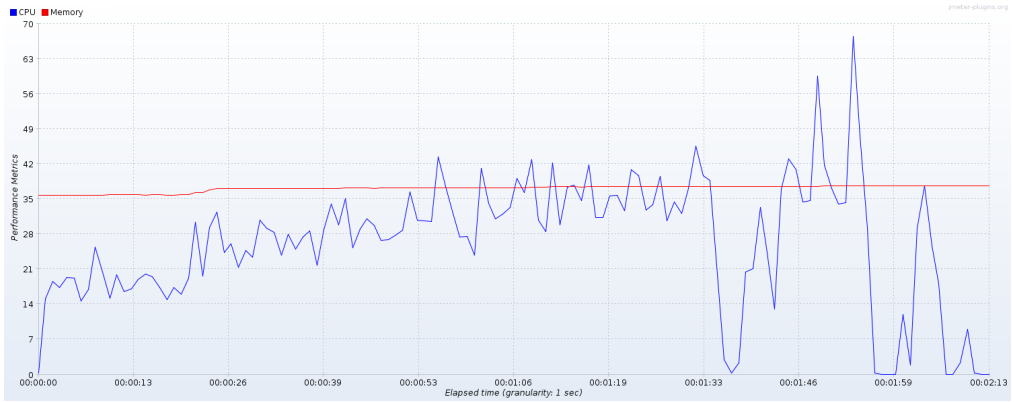


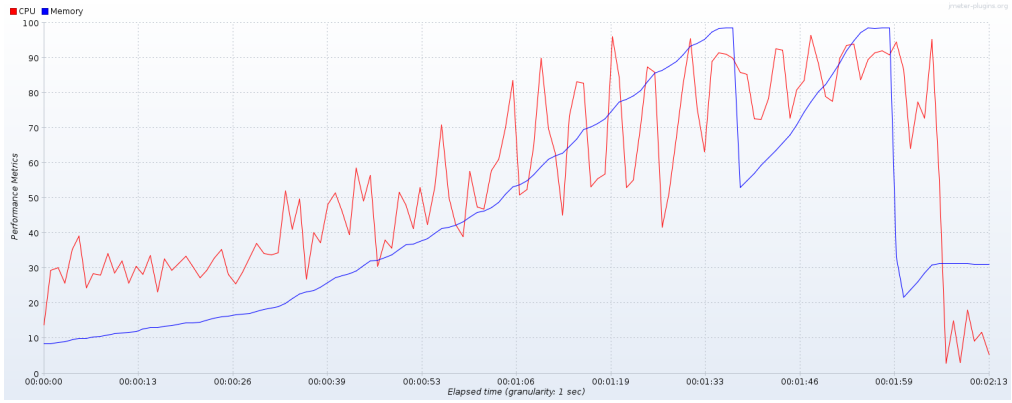
Figure 16. Single processor: CPU and RAM memory utilization reported by the "Send alert" task.

In Figure 17a, we show the resource utilization reported by Jayma with a distributed deployment. MongoDB presents average utilization levels of 40%. There are two CPU peaks close to 70% during the last time interval due to the increasing workload. In Figure 17b, we show that Jayma requires a larger amount of RAM memory, reaching utilization levels close to 100% in the last time interval. The CPU also reports high levels of utilization above 90%. Therefore, in the Jayma application, both resources (CPU and memory) tend to be critical when the application is overloaded. Finally, Figure 18 shows the communication cost for the task "Report User Status". As expected, the communication cost tends to increase as we increase the number of requests to be processed every 30 s.



(a)

Figure 17. *Cont.*



(**b**)

Figure 17. Distributed deployment: Resource utilization reported by (**a**) MongoDB and (**b**) the workers for the task "Send alert".

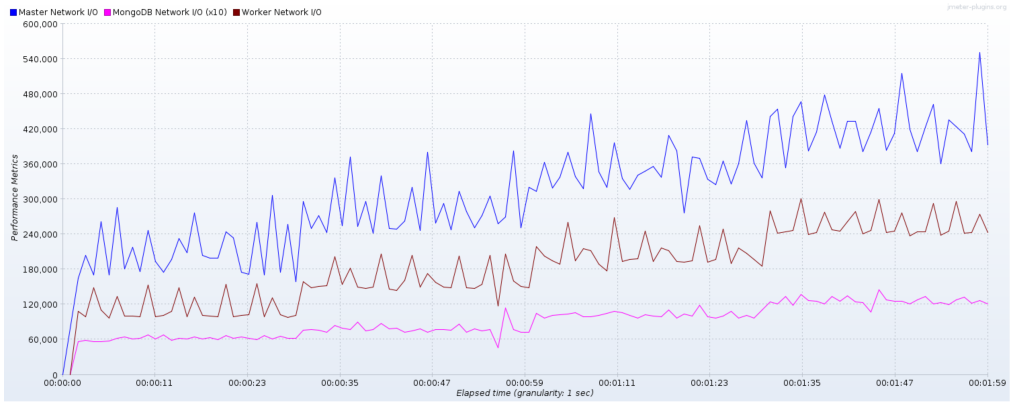


Figure 18. Communication cost in bytes reported by the Jayma application.

5. Related Work

In this section, we first present some discussion about software applications used to help, assist affected people, and manage resources after a natural disaster occurs. Then, we discuss previous work using bot systems aimed to inform and to aid people.

The evaluation and prediction of the outcome of natural disasters have been extensively studied in the technical literature. The studies presented in [22–25] use agent-based simulation tools for population modeling, infrastructure development planning, country reconstruction humanitarian support operations, and simulation of rivers and overflows, among others. These studies can be used by people involved in planning rescue activities.

A study performed after the tsunami in Tailandia 2004 [26] presents the problems detected in the public management of natural disasters such as logistic problems for distributing goods for emergency relief; lack of effective collaboration among institutions in different levels; lack of encouragement for participation of local and international non-governmental organizations (NGOs); lack of education and knowledge for tsunami in potential disaster-affected communities and lack of information management or database system. The authors present the use cases of an application to solve logistic problems for distributing goods for emergency relief and to coordinate help between different institutions and NGOs. They conclude that NGOs must coordinate the tasks during a disaster in addition to sharing vital information with each other during effective communication channels.

As communication networks are a critical resource during a natural disaster, the work in [27] provides guidelines to support fault tolerance with different communication networks such as wifi, SDR, Satellite Telephony, etc., using information obtained during Hurricane Katrina. The authors propose using a relational database and a front-end deployed on a grid computing to support a high load of requests in a short time. The

author shows that the fastest recovering network is in data networks. This supports our proposal of using bot-based applications.

Different software applications—without bots running on social platforms such as Messenger of Facebook or Telegram—have been presented in the technical literature [28–33]. The system presented in [28] was initially developed in 2–3 weeks in response to the tsunami in Sri Lanka (2004) by volunteer developers. Today, it has a series of libraries and APIs: Location API, GIS API, generic Reporting API, DataBase, and Synchronization API, among others. It supports different operations such as the registration of organizations, the request management system, and the register of persons. However, the system is deployed on a single processor, and the authors do not present a performance evaluation. Therefore, its proper deployment during a natural disaster requires engineers who are experts in fault tolerance, saturated systems, and managing peaks of requests.

A mobile application presented in [29] takes advantage of probabilistic geosocial information collected before the event to guide an earthquake victim and rescuers during and after the earthquake. It has four operation modes. The standby mode is used to collect data about users, their regularly visited locations, and their social relationships. Another mode, alert mode, warns of an imminent earthquake based on data collected from a variety of sensors and government alert systems. In disaster mode, it helps users cope with their tasks, such as the evacuation and rescue of victims buried under the rubble of collapsed buildings. In recovery mode, it facilitates family reunification, provides information on help centers, and sends warnings about the spread of disease. The authors present the use cases for the application without validation. Moreover, they do not provide detail about the architecture, the components, or the performance of the application. The work in [30] presents a prototype of a spontaneous voluntary coordination app. It uses PostgreSQL as the webserver. However, it has not been tested on a large scale. The authors present results based on the simulation of 200,000 agents, but they do not report information about how the application behaves under a high load of user requests, fault-tolerant deployment, scalability, or possible bottlenecks. Similar to [30], the authors in [31] present an application to organize pre-qualified volunteers (e.g., volunteers with medical skills). The application allows enrolling and characterizing volunteers, and through a survey, the application predicts the willingness of users to participate in a rescue of which they are notified by a smartphone application. As in the previous case, the paper does not give information about the architecture of the application, implementation, or about the performance evaluation.

There are applications for emergency management designed for P2P networks [4,32]. However, managing a P2P network requires implementing a distributed hash table (DHT) and communication protocols such as Pastry or Chord. Moreover, high peaks of workloads are difficult to manage, tending to saturate not only computation resources but also communication channels as messages travel from one peer to another until they reach their destination.

A classification model is used in [33] as part of a conceptual framework for the development of emergency applications. The authors evaluate the proposal using a model to select, classify, and prioritize tweet messages obtained from the Mexico 2017 earthquake for rescue operations. The authors do not report about the training time and about the execution time of the model. Moreover, the ability of the model to fit other datasets is not discussed.

Some applications collect data from real-time sensors and drones [34,35]. Sensors have been used in fixed [36–38] and mobile sensor networks [39,40] for environmental monitoring and for planning rescue operations. In particular, drones have been proven to be a useful tool to obtain a map to support searching in a post-disaster scenario [41]. In the same line, the work presented in [42] simulates disaster scenarios and analyzes data by using the unmanned aerial vehicle technology +BIM model's monomer and visual information query. The authors propose using these technologies to look at the disaster data for planning and early warning. However, these kinds of applications do not include the participation of volunteers, the managing of rescue tasks, or the communication with victims.

Nowadays, bot-based software applications have been widely used in different areas such as journalism [43], where bots are used to personalize the information given to users. A user interacts with the bots, which delivers increasingly specific information through the use of questionnaires and machine learning techniques. These bots are typically implemented on Twitter, Facebook, and Telegram platforms. Bots are also used in the supply chain process [44] to analyze the user queries and provide information on the queried orders and supplies; they are also used in e-commerce and retail companies [45,46] where users can register, and then the bot shows catalogs, guides, shopping carts, reviews, tracks the purchases, and generates purchase orders; for medical assistance [11,47], virtual robots represented in a bot are used to communicate with patients and assist them in their routine needs; and they are also used for data collection [13,48,49]. In particular, the bot presented in [48] is designed to collect and validate real-time crowd-sourced flood users reports via social media (Messenger of Facebook bot or Twitter) to enhance the city's resilience to extreme weather events. The system is deployed on Amazon Web Server (AWS) to support fault tolerance and scalability. However, none of these applications require processing a large amount of user requests as fast as possible without saturating the available resources.

As we explained before, bots are typically used to collect data from different users. In the context of crisis and disaster management, the authors in [13] present a framework for a disaster-related information retrieval system and immediate notifications to support the execution of mine safety procedures. The system utilizes instant messaging (IM) applications as the user interface to look up information and send messages to announce the occurrence of disaster events. Following the same idea of using bots to collect data, the authors in [49] present their experience of applying a bot-based system for the government of Taiwan for disaster response operations. In particular, they describe a framework for implementing a bot as the data management system. The bot is designed to answer questions containing keywords from a given list. If the user forgets the keywords, the application will not allow them to advance or give correct information. The paper evaluates the usability of the application by checking if people were able to finish the tasks. Bots can be also used to provide information after a natural disaster strikes. This is the case of the work presented in [50], which allows sharing videos with subtitles and sign language for deaf people to help them during the evacuations. The authors report a usability satisfaction level of 95% in deaf people and 60% in hearing people. In addition, the authors in [51] present a bot using Telegram to provide emergency information to foreign people in Japan. It includes evacuation information and real-time disaster information, and the real-time disaster information is based on the user's current location. People can also share disaster-related pictures via the proposed application. People can share images of the disaster; however, it does not take into account, for example, whether the photos are real or if they are analyzed by experts.

In Table 4, we compare the main features presented by Ayni, Rimay, Jayma, and other bot-based applications designed for crisis situations such as natural disasters. The table summarizes which applications support fault tolerance (Fault Tol.), scalability (Scal.), if the authors present a performance evaluation taking into account botlenecks, critical tasks, and hardware resources (Perf.) and a usability evaluation (Usab.) of applications. The column Validation shows (A) if the experiments were performed on a real system, (B) with simulations and/or (D) with synthetic data. (C) indicates whether the application was validated by experts. In the column named Type, we use the number (1) for previous work presenting a guideline, (2) for applications or simulators that have been implemented, and (3) for previous work presenting a model or proposal not actually implemented. Finally, the column Deployment indicates if the applications can be executed sequentially, in parallel, and/or distributed or if it is not specified.

Notice that most of the previous works describe the application proposed, but they do not present experiments to evaluate their features. Moreover, previous works are typically focused on specific natural disasters and are not re-used later. In this paper, we evaluate the performance achieved by Ayni, Rimay, and Jayma, which also support fault tolerance,

scalability, and can be executed in parallel and distributed data centers due to being built using common practice for distributed software architecture design. In particular, we use Dockers and Kubernetes technologies. Additionally, we execute the experiments on real systems, and we validate the tasks executed by each application based on expert consensus. To the best of our knowledge, no previous work studies the behavior and performance achieved by applications designed for natural disasters situations running on commodity hardware usually available from public organizations such as universities.

Table 4. Main features presented by Ayni, Rimay, Jayma, and other applications presented in the technical literature. In the column Validation, we use the letter (A) if the experiments were performed on a real system, (B) with simulations and/or (D) with synthetic data. (C) indicates whether the application was validated by experts. In the column Type, we use the number (1) to indicate that a guideline is presented, (2) for applications or simulators that have been implemented, and (3) for papers presenting a model or proposal not actually implemented.

App/Ref.	Perf.	Usab.	Scal.	Fault Tol.	Validation	Type	Deployment
Rimay	Yes	–	Yes	Yes	(A), (C)	(2)	Parall./Distrib.
Ayni	Yes	–	Yes	Yes	(A), (C)	(2)	Parall./Distrib.
Jayma	Yes	–	Yes	Yes	(A), (C)	(2)	Parall./Distrib.
[13]	–	–	–	Yes	(A)	(2)	Distrib.
[22–24]	–	–	–	–	(B), (C)	(2)	Sequential
[25]	–	–	–	–	(B), (D)	(2)	Sequential
[27]	–	–	–	Yes	–	(3)	Distrib.
[28]	–	–	–	–	(C)	(2)	Sequential
[29]	–	–	–	–	–	(3)	Not Specified
[30]	Yes	Yes	–	–	(B)	(2)	Not Specified
[31]	–	–	–	–	–	(3)	Sequential
[32]	–	–	–	–	–	(2)	Distrib.
[33]	–	–	–	–	(B)	(1), (2)	Not Specified
[48]	–	–	Yes	Yes	–	(2)	Sequential
[49]	–	Yes	–	–	(C)	(1), (2)	Sequential
[50]	–	Yes	–	–	(C)	(2)	Sequential
[51]	–	–	–	–	–	(2)	Sequential

6. Discussion

The bots designed for Jayma, Ayni, and Rimay run on third-party platforms such as Messenger of Facebook and Telegram. These platforms apply some restrictions to prevent malicious uses and an inefficient user experience. In other words, they restrict the development and the execution of our applications as follows.

The bots of Ayni and Rimay, which use Telegram (https://core.telegram.org/bots/faq, accessed on 28 January 2022), are implemented with https. Telegram supports the upload of files up to 50 Mb each, and the maximum download of each file is 20 Mb. Additionally, we can send up to 30 messages per second to different users and 20 messages per second to the same group. This is a very strong restriction, as it limits the number of users we can reach per second during an emergency situation.

The bot of Jayma uses the Messenger (https://developers.facebook.com/docs/graph-api/overview/rate-limiting, accessed on 28 January 2022) of Facebook, which allows each user to send up to 250 messages per second to different recipients. However, it limits the number messages a given user can send to a given recipient per hour to 200. It supports a maximum upload of 25 Mb per file and a maximum of 2000 characters per message.

Therefore, the applications presented in this work are designed to meet the restrictions and recommendations of the external platform. However, if Messenger or Telegram fail,

our applications will not be able to run. On the other hand, we choose to rely on these platforms because in case of emergency situations, people tend to use applications they are familiar with rather than to download, install, and learn how to use new applications, as shown in [7,8]. A comparative analysis of the use of applications for communication via messages compares WhatsApp, Viber, and Telegram in [8]. Despite the fact that Viber and Telegram presented better security properties, WhatsApp widely exceeds the number of users worldwide with at least 60% of users, this due to its popularity, ease of use, and its current support of Facebook. A study [7] with 4000 people showed that 70% use Facebook and 24% use Twitter. Facebook seems to be especially attractive to mobile users. Internet skills turn out to not be significant for the choice of use of Facebook, but it is only the mere choice of wanting to use it. Facebook use is influenced by age and gender. For Twitter, age and income influence its use but not gender and education.

7. Conclusions and Future Work

In this work, we evaluate the feasibility of using bots in three applications to collect the information of affected people and also to register spontaneous volunteers and to coordinate the tasks executed in different missions. In particular, we present a performance evaluation to detect the critical tasks and saturation points of the bot-based software applications. The first application called Jayma is designed to send a message to the contacts of the affected people with information about their status and location. The second application is designed to manage a set of events and tasks created by administrators and to coordinate volunteers who want to help. The third application, Ayni, is aimed at registering volunteers and managing emergency campaigns and tasks. The bots of these applications run on third-party platforms such as Messenger of Facebook and Telegram. This limits the number of messages we can send per second to the users. In addition, if these platforms fail, our applications will not be available. However, these are well-known platforms, and a person who can potentially participate as a spontaneous volunteer will be familiar with them and will not need to install any additional software.

We implement the applications as stateless and deploy them on a microservices-based platform with Kubernetes and Docker technologies. Therefore, the applications can be deployed and executed in parallel due to its replication capabilities and also on distributed environments. The platform architecture contains a front-end, a back-end, a bot-back-end, and a repository (database). We instrument the codes to obtain benchmarks of the tasks executed in each application that run on different resources.

We show that the applications are able to cope with critical mission requirements. This fact is shown throughout a comprehensive performance evaluation study on actual commodity hardware. To this end, we conduct experiments to analyze and evaluate potential bottlenecks and determine capacity in terms of the number of requests served per second in each of the applications. The validation of the tasks executed by the applications was based on expert consensus. The experiments on a single processor showed that the Ayni application supports less than 60 requests per second without crashing. With a higher number of requests per second, some tasks became unstable, and some requests did not complete. In this application, the CPU tends to saturate faster than other resources. In the Rimay application, the CPU is also a critical resource showing utilization levels close to 70%. Meanwhile, in the Jayma application, the RAM memory is the most critical resource, showing utilization levels close to 100%.

The results show that it is possible to build a software stack designed to support the execution of software applications that have properties for emergencies such as high availability, elasticity, fault tolerance, scalability, mobility (change data center dynamically), and multi-environment. All these features have not been included and evaluated in previous bot-based software applications. The multi-environment is satisfied by using bots of social media applications commonly used by people plus the use of Dockers that allow deployment on different operating systems and facilitate deployment in the cloud. Mobility is achieved by using Dockers and Kubernetes for application management and monitoring.

Fault tolerance is achieved by using Kubernetes and Docker registry (Container Database), data replication, and volatile applications where data do not need to be recovered after a failure. Scalability is achieved by using Kubernetes. A configuration file allows generating replicas of the application and also stateless applications design.

As future work, we plan to include new communication channels between the users and other agents rather than friends or family to prevent disaster events that can propagate to other cities or communities such as blackouts or floods. We also plan to re-design the bots of the applications so they can be executed without third-party platforms such as Telegram. However, this will require disseminating the applications and training the population so that they will be prepared to use them. Finally, we plan to develop a capacity-planning methodology to automatically determine the number of replicas of each component of the applications to then efficiently deploy them in data centers provided by public organizations.

Author Contributions: Investigation, G.O.-L., L.V.-C., V.G.-C. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by the Chilean Agency for Research and Development (ANID) under grant Basal Centre CeBiB code FB0001. Gabriel Ovando-Leon was funded by the National Agency for Research and Development (ANID), Scholarship Program, Doctorado Becas Chile 2017-21171745.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Wang, C.; Du, W.; Chen, Z.; Chen, N.; Wang, W. An Event Modeling Software for Natural Disasters: Design and Implementation. In Proceedings of the International Conference on Geoinformatics, Kunming, China, 28–30 June 2018; pp. 1–4.
2. Segura, A.; Olmedo, G.; Acosta, F.; Santillán, M. Designing a system for monitoring and broadcasting early warning signs of natural disasters for Digital Terrestrial Television. In Proceedings of the IEEE Latin-American Conference on Communications (LATINCOM), Arequipa, Peru, 4–6 November 2015; pp. 1–6.
3. Mas Machuca, C.; Secci, S.; Vizarreta, P.; Kuipers, F.; Gouglidis, A.; Hutchison, D.; Jouet, S.; Pezaros, D.; Elmokashfi, A.; Heegaard, P.; et al. Technology-related disasters: A survey towards disaster-resilient Software Defined Networks. In Proceedings of the International Workshop on Resilient Networks Design and Modeling (RNDM), Halmstad, Sweden, 13–15 September 2016, pp. 35–42.
4. Loor, F.; Manriquez, M.; Gil-Costa, V.; Marin, M. Feasibility of P2P-STB based crowdsourcing to speed-up photo classification for natural disasters. *Cluster Comput.* **2021**, *25*, 279–302. [CrossRef]
5. Wu, X.; Mazurowski, M.; Chen, Z.; Meratnia, N. Emergency message dissemination system for smartphones during natural disasters. In Proceedings of the International Conference on ITS Telecommunications, St. Petersburg, Russia, 23–25 August 2011, pp. 258–263.
6. Erdelj, M.; Natalizio, E. Drones, smartphones and sensors to face natural disasters. In Proceedings of the ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications, Munich, Germany, 10–15 June 2018; pp. 75–86.
7. Blank, G.; Lutz, C. Representativeness of Social Media in Great Britain: Investigating Facebook, LinkedIn, Twitter, Pinterest, Google+, and Instagram. *Am. Behav. Sci.* **2017**, *61*, 741–756. [CrossRef]
8. Sutikno, T.; Handayani, L.; Stiawan, D.; Riyadi, M.A.; Subroto, I.M.I. WhatsApp, viber and telegram: Which is the best for instant messaging? *Int. J. Electr. Comput. Eng.* **2016**, *6*, 909.
9. Kryvasheyeu, Y.; Chen, H.; Obradovich, N.; Moro, E.; Van Hentenryck, P.; Fowler, J.; Cebrian, M. Rapid assessment of disaster damage using social media activity. *Sci. Adv.* **2016**, *2*, e1500779. [CrossRef]
10. Lu, Y.; Hu, X.; Wang, F.; Kumar, S.; Liu, H.; Maciejewski, R. Visualizing social media sentiment in disaster scenarios. In Proceedings of the International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 1211–1215.
11. Battineni, G.; Chintalapudi, N.; Amenta, F. AI Chatbot Design during an Epidemic like the Novel Coronavirus. *Healthcare* **2020**, *8*, 154. [CrossRef]
12. Syed, H.A.; Schorch, M.; Pipek, V. Disaster Learning Aid: A Chatbot Centric Approach for Improved Organizational Disaster Resilience. In *Information Systems for Crisis Response and Management Conference*; Virginia Tech: Blacksburg, VA, USA, 2020; pp. 448–457.

13. Tsai, M.H.; Chan, H.Y.; Chan, Y.L.; Shen, H.K.; Lin, P.Y.; Hsu, C.W. A Chatbot System to Support Mine Safety Procedures during Natural Disasters. *Sustainability* **2021**, *13*, 654. [CrossRef]
14. Bradshaw, S.; Brazil, E.; Chodorow, K. *Mongodb: The Definitive Guide: Powerful and Scalable Data Storage*; O'Reilly Media: Sebastopol, CA, USA, 2019.
15. Lakshman, A.; Malik, P. Cassandra: A decentralized structured storage system. *Oper. Syst. Rev.* **2010**, *44*, 35–40. [CrossRef]
16. Osborne, G.; Weninger, T. Ozy: A General Orchestration Container. In Proceedings of the International Conference on Web Services (ICWS), San Francisco, CA, USA, 27 June–2 July 2016; pp. 609–616.
17. Adufu, T.; Choi, J.; Kim, Y. Is container-based technology a winner for high performance scientific applications? In Proceedings of the Asia-Pacific Network Operations and Management Symposium (APNOMS), Busan, Korea, 19–21 August 2015; pp. 507–510.
18. Burns, B.; Beda, J.; Hightower, K. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*; O'Reilly Media: Sebastopol, CA, USA, 2019.
19. Merkel, D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux* **2014**, *2014*, 2.
20. DeJonghe, D. *Nginx CookBook*; O'Reilly Media: Sebastopol, CA, USA, 2020.
21. Erinle, B. *Performance Testing with JMeter 2.9*; Packt Ltd.: Birmingham, UK, 2013.
22. Massei, M.; Poggi, S.; Agresta, M.; Ferrando, A. Development planning based on interoperable agent driven simulation. *Comput. Sci.* **2014**, *5*, 395–407. [CrossRef]
23. Merkuryeva, G.; Merkuryev, Y.; Sokolov, B.V.; Potryasaev, S.; Zelentsov, V.A.; Lektauers, A. Advanced river flood monitoring, modelling and forecasting. *Comput. Sci.* **2015**, *10*, 77–85. [CrossRef]
24. Barnes, B.; Dunn, S.; Pearson, C.; Wilkinson, S. Improving human behaviour in macroscale city evacuation agent-based simulation. *Disaster Risk Reduct.* **2021**, *60*, 102289. [CrossRef]
25. Wang, Z.; Jia, G. A novel agent-based model for tsunami evacuation simulation and risk assessment. *Nat. Hazards* **2021**, *105*, 2045–2071. [CrossRef]
26. Moe, T.L.; Pathranarakul, P. An integrated approach to natural disaster management. *Disaster Prev. Manag.* **2006**, *15*, 396–413.
27. Banipal, K. Strategic approach to disaster management: Lessons learned from Hurricane Katrina. *Disaster Prev. Manag.* **2006**, *15*, 484–494. [CrossRef]
28. Careem, M.; De Silva, C.; De Silva, R.; Raschid, L.; Weerawarana, S. Sahana: Overview of a Disaster Management System. In Proceedings of the 2006 International Conference on Information and Automation, Colombo, Sri Lanka, 15–17 December 2006; pp. 361–366.
29. Bekhor, S.; Cohen, S.; Doytsher, Y.; Kanza, Y.; Sagiv, Y. A Personalized GeoSocial App for Surviving an Earthquake. In Proceedings of the ACM SIGSPATIAL International Workshop on the Use of GIS in Emergency Management, Bellevue, WC, USA, 3–6 November 2015; pp. 1–6.
30. Betke, H. A Volunteer Coordination System Approach for Crisis Committees. In Proceedings of the International Conference on Information Systems for Crisis Response and Management, Rochester, NY, USA, 20–23 May 2018; pp. 786–795.
31. Horstmann, A.C.; Winter, S.; Rösner, L.; Krämer, N.C. S.O.S. on my phone: An analysis of motives and incentives for participation in smartphone-based volunteering. *Contingencies Crisis Manag.* **2018**, *26*, 193–199. [CrossRef]
32. Catarci, T.; de Rosa, F.; de Leoni, M.; Mecella, M.; Angelaccio, M.; Dustdar, S.; Gonzalvez, B.; Iiritano, G.; Krek, A.; Vetere, G.; et al. WORKPAD: 2-Layered Peer-to-Peer for Emergency Management through Adaptive Processes. In Proceedings of the Collaborative Computing: Networking, Applications and Worksharing, Atlanta, GA, USA, 17–20 November 2006; pp. 1–9.
33. Freitas, D.P.; Borges, M.R.S.; de Carvalho, P.V.R. A conceptual framework for developing solutions that organise social media information for emergency response teams. *Behav. Inf. Technol.* **2020**, *39*, 360–378. [CrossRef]
34. Qian, K.; Claudel, C. Real-time mobile sensor management framework for city-scale environmental monitoring. *Comput. Sci.* **2020**, *45*, 101205. [CrossRef]
35. Alfeo, A.L.; Cimino, M.G.; De Francesco, N.; Lega, M.; Vaglini, G. Design and simulation of the emergent behavior of small drones swarming for distributed target localization. *Comput. Sci.* **2018**, *29*, 19–33. [CrossRef]
36. Kerkez, B.; Glaser, S.D.; Bales, R.C.; Meadows, M.W. Design and performance of a wireless sensor network for catchment-scale snow and soil moisture measurements. *Water Resour. Res.* **2012**, *48*. [CrossRef]
37. Weimer, J.E.; Sinopoli, B.; Krogh, B.H. A relaxation approach to dynamic sensor selection in large-scale wireless networks. In Proceedings of the International Conference on Distributed Computing Systems Workshops, Beijing, China, 17–20 June 2008; pp. 501–506.
38. Krause, A.; Singh, A.; Guestrin, C. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Mach. Learn. Res.* **2008**, *9*, 235–284.
39. Tokekar, P.; Branson, E.; Vander Hook, J.; Isler, V. Tracking aquatic invaders: Autonomous robots for monitoring invasive fish. *IEEE Robot. Autom. Mag.* **2013**, *20*, 33–41. [CrossRef]
40. Tokekar, P.; Vander Hook, J.; Mulla, D.; Isler, V. Sensor planning for a symbiotic UAV and UGV system for precision agriculture. *IEEE Trans. Robot.* **2016**, *32*, 1498–1511. [CrossRef]
41. Whitehead, K.; Hugenholtz, C.H. Remote sensing of the environment with small unmanned aircraft systems (UASs), part 1: A review of progress and challenges. *Unmanned Veh. Syst.* **2014**, *2*, 69–85. [CrossRef]
42. Wu, B.; Fu, R.; Chen, J.; Zhu, J.; Gao, R. Research on Natural Disaster Early Warning System Based on UAV Technology. In *IOP Conference Series: Earth and Environmental Science*; IOP Publishing: Chengdu, China, 2021; Volume 787, pp. 1–8.

43. Jones, B.; Jones, R. Public service chatbots: Automating conversation with BBC News. *Digit. J.* **2019**, *7*, 1032–1053. [CrossRef]
44. Angelov, S.; Lazarova, M. E-Commerce Distributed Chatbot System. In Proceedings of the Balkan Conference on Informatics, Sofia, Bulgaria, 26–28 September 2019; pp. 1–8.
45. Licapa-Rodriguez, R.; Gomez-Ramos, J.; Mauricio, D. EcoBot: Virtual assistant for e-commerce of ecological bricks based on Facebook Messenger. In Proceedings of the IEEE Engineering International Research Conference (EIRCON), Lima, Peru, 27–29 October 2021; pp. 1–4.
46. Tran, A.D.; Pallant, J.I.; Johnson, L.W. Exploring the impact of chatbots on consumer sentiment and expectations in retail. *Retail. Consum. Serv.* **2021**, *63*, 102718. [CrossRef]
47. Fadhlallah, G.M. A Deep Learning-Based Approach for Chatbot: Medical Assistance a Case Study. Master's Thesis, Facultédes Lettres et des Langues FLL, Biskra, Argelia, 2021.
48. Dharmapuri Sridhar, M.P. Real-Time Flood Mapping for Disaster Management Decision Support in Chennai. Master's Thesis, System Design and Management Program, MIT, Cambridge, MA, USA, 2017.
49. Tsai, M.H.; Yang, C.H.; Chen, J.Y.; Kang, S.C. Four-Stage Framework for Implementing a Chatbot System in Disaster Emergency Operation Data Management: A Flood Disaster Management Case Study. *KSCE Civ. Eng.* **2021**, *25*, 503–515. [CrossRef]
50. Rotondi, L.; Zuddas, M.; Marsella, P.; Rosati, P. A Facebook Page Created Soon After the Amatrice Earthquake for Deaf Adults and Children, Families, and Caregivers Provides an Easy Communication Tool and Social Satisfaction in Maxi-Emergencies. *Prehosp. Disaster Med.* **2019**, *34*, 137–141. [CrossRef]
51. Ahmady, S.E.; Uchida, O. Telegram-Based Chatbot Application for Foreign People in Japan to Share Disaster-Related Information in Real-Time. In Proceedings of the Conference on Computer and Communication Systems (ICCCS), Shanghai, China, 15–18 May 2020; pp. 177–181.