DiFFuSE, a Distributed Framework for Cloud-based Epidemic Simulations: a Case Study in Modelling the Spread of Bovine Viral Diarrhea Virus

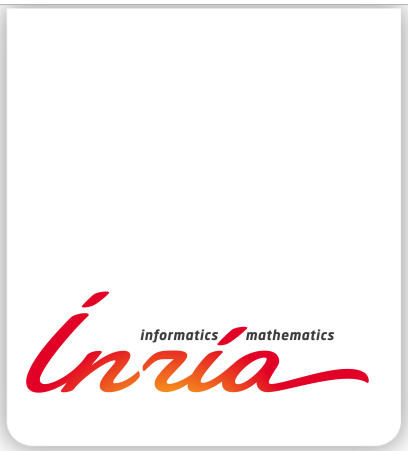
Linh Manh Pham, Nikos Parlavantzas, Christine Morin, Sandie Arnoux, Luyuan Qi, Philippe Gontier, Pauline Ezanno

DiFFuSE, a Distributed Framework for Cloud-based Epidemic Simulations: a Case Study in Modelling the Spread of Bovine Viral Diarrhea Virus

Linh Manh Pham , Nikos Parlavantzas, Christine Morin, Sandie Arnoux, Luyuan Qi, Philippe Gontier, Pauline Ezanno

RESEARCH REPORT
N° 9094
July 2017
Project-Team MYRIADS

ISSN 0249-6399 ISRN INRIA/RR--9094--FR+ENG



DiFFuSE, a Distributed Framework for Cloud-based Epidemic Simulations: a Case Study in Modelling the Spread of Bovine Viral Diarrhea Virus

Linh Manh Pham *, Nikos Parlavantzas*, Christine Morin*, Sandie Arnoux†, Luyuan Qi†, Philippe Gontier†, Pauline Ezanno†

Project-Team MYRIADS

Research Report n° 9094 — July 2017 — 18 pages

Abstract: Performing large-scale epidemic simulations is time-consuming and requires massive amounts of computing resources. An attractive option for providing such resources is cloud computing. However, fully exploiting cloud resources requires converting existing simulation applications into elastic, cloud-enabled applications. To this end, this paper proposes a novel service-based framework for building and executing epidemic simulations in the cloud and describes how the framework was applied to build a cloud-enabled simulation of the spread of the bovine viral diarrhea virus. The report provides experimental evidence that the framework increases performance and allows exploring cost-performance trade-offs while automatically handling failures and supporting elastic allocation of resources from multiple clouds.

Key-words: simulation, cloud computing

* INRIA Rennes-Bretagne Atlantique, Rennes, France
† BIOEPAR, INRA, Oniris

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

DiFFuSE, un Framework Distribué pour les Simulations Épidémiques Basées sur le Cloud: une Étude de Cas de la Modélisation de la Propagation du Virus de la Diarrhée Virale Bovine

Résumé : La réalisation des simulations épidémiques à grande échelle exige beaucoup de temps et des quantités de ressources informatiques massives. Une option attrayante pour fournir de telles ressources est le cloud computing. Cependant, l'exploitation des ressources de cloud nécessite la conversion des applications de simulation existantes en applications élastiques basées sur le cloud. À cette fin, cet article propose un nouveau framework basé sur les services pour la construction et l'exécution de simulations épidémiques sur le cloud et décrit comment ce framework a été appliqué pour construire une simulation de la propagation du virus de la diarrhée virale bovine. Le rapport fournit des résultats expérimentaux qui indiquent que le framework augmente les performances et permet d'explorer les compromis coût-performance tout en gérant automatiquement les pannes et en soutenant l'allocation élastique des ressources de plusieurs clouds.

Mots-clés : simulation, nuages informatiques

1 INTRODUCTION

One of the main concerns in our connected world is the spread of infectious diseases on both humans and animals. Governments and researchers invest considerable time and effort to study the spread of such diseases in order to develop strategies for controlling them. A powerful tool for studying pathogen spread is mechanistic modelling, integrating the relevant biological knowledge to represent the system either using a mathematical or computational framework. To predict pathogen spread in a particular area, it can be relevant to use a data-driven framework for part of the processes, exploiting real-life observed data as inputs. Such data barely concerns directly the epidemiological processes but rather processes related to the population dynamics of hosts, which are often very well documented; this is the case, e.g., with cattle in Europe with the individual follow-up of each animal from birth to death and with the record of all events that occurred (trade movements, calving, etc.) [1]. Such data enables modellers to build very precise and realistic models adapted to territorial characteristics [2, 3].

Running epidemic simulation models can take a large amount of time, especially when using fine-grained models considering different scales (individual, population, meta-population) and also when accounting for additional layouts (e.g., economic aspects such as farm management decisions) corresponding to real-life systems [4]. Moreover, to better understand the biological system through the analysis of the model behaviour, many scenarios generally have to be compared, e.g., through a model sensitivity analysis. In addition, as biological systems are highly variable, stochasticity often is accounted for, giving rise to intensive simulations to predict accurate results.

To reduce the execution time of large-scale simulations, parallel computing can be used. Parallelisation can exist at the single-computer level using multiple cores and processors as well as the distributed level using multiple computers interconnected through a network. Epidemic simulations can exploit parallelisation by executing multiple simulation runs in parallel. Nevertheless, executing enough runs to produce reliable results for stochastic models often demands massive amounts of computing resources. A popular solution for obtaining such resources is using high-end multicore systems or clusters. Such infrastructures are typically shared among local users, resulting in high contention and long waiting times. Extending such infrastructures through investing in new dedicated hardware is often not economically efficient.

Another widely-used solution for obtaining computing resources is grid computing, which allows sharing computers from multiple clusters in multiple sites. A major limitation of grids is the heterogeneity of the software stacks in each cluster, which makes application deployment difficult and potentially creates inconsistent application outputs [5]. Moreover, grid-based applications are assigned a statically-fixed number of resources without taking into account real resource demand after application deployment. This leads to poor resource utilization or reduced performance in the face of dynamic variations in resource demand.

Cloud computing has emerged as a popular approach to overcome the limitations of grid computing. Owing to virtualization, each cloud application has its own software stack, independently of the physical infrastructure. Cloud users elastically obtain and release resources on demand using a pay-as-you-go model, whereby they are only charged for their actual resource usage. Nevertheless, converting complex scientific applications, such as epidemic simulations, to take advantage of clouds presents a number of challenges.

First, legacy simulation applications are most often sequential, monolithic applications designed to be run on a single node. Taking advantage of clouds requires considerable manual effort for obtaining cloud resources, configuring and launching multiple runs of the simulation on these resources, making input data available to these runs, and collecting and processing simulation results. The amount of effort is increased when the simulation application is to be deployed on multiple clouds, which is useful for avoiding dependence on a single vendor, enhancing availability, and taking advantage of lower resource prices or specialized resource capabilities. Second, to optimize resource utilization and performance, simulation applications may need to dynamically scale up and down their resource usage. For example, the simulation initialization phase may require less compute resources than the main processing phase, thus making it more efficient to dynamically increase the allocation of compute resources. Clouds provide elastic resources, but adding these resources to a running simulation requires automated support. Third, when executing long-running, distributed applications, such as epidemic simulations, on the cloud, failures are unavoidable. Automated support is needed to handle failures, avoiding data loss and ensuring the correct and efficient completion of the simulation.

To address these challenges, we have developed a framework, called DiFFuSE, which enables epidemic simulation applications to take full advantage of cloud environments. Specifically, the paper makes two main contributions. First, it describes the DiFFuSE framework for building and executing epidemic simulations. The main novelty of DiFFuSE lies in its service-based architecture; simulation applications are decomposed into services that can be deployed across multiple clouds and independently scaled, while the framework transparently handles service failures. Second, the paper presents the application of DiFFuSE to building a cloud-enabled simulation of the spread of bovine viral diarrhea virus (BVDV) and uses this application to evaluate the benefits of the DiFFuSE framework.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 briefly presents the BVDV case study, the epidemic simulation used throughout the paper as a practical example. Section 4 details the architecture of DiFFuSE and its application in the BVDV case study. An extensive set of experiments is reported in Section 5 using the BVDV example. Finally, Section 6 concludes the paper.

2 RELATED WORK

Much research work has focused on supporting the execution of scientific applications demanding large amounts of computing power. In the following, we consider work that specifically targets the execution of epidemic simulations.

[6] presents a cloud-based approach for performing simulations of pandemic influenza. The approach relies on the Condor architecture in which a central master schedules and distributes jobs and a set of workers receive and execute the jobs. The approach allows adding workers by renting Amazon EC2 virtual machines (VMs). However, the tests show that the network capacity of the master node can become a bottleneck when the number of workers increases.

[4] explores the use of OpenMP to parallelise pandemic simulations. The goal is to increase performance by taking advantage of multi-core processors. The evaluation demonstrates that the performance can be further improved by dynamically switching between single- and multi-core modes as the computational load varies. This work does not consider parallelisation at the distributed level.

[7] proposes a cloud-based framework for simulating the spread of epidemic diseases. The framework applies loop decomposition to parallelise the simulation on multiple VMs of a private cloud, built using the Xen Cloud Platform. This framework is restricted to a specific epidemic model, namely, a modified version of the SEIR model (Susceptible, Exposed, Infectious, Recovered).

[8] presents an approach for executing a compute-intensive application for epidemic analysis on cloud resources. The approach relies on an iterative development approach and uses the Nimbus cloud to obtain resources on demand. This work does not consider the cost of using cloud resources.

Epidemic simulations can be structured as bag-of-task or master-worker applications, in which a set of independent simulation runs are executed in parallel. The following work is thus also related to ours. [9] presents a framework, called BOT, for developing bag-of-tasks scientific applications on the mOSAIC cloud platform. Similarly to our work, BOT can be used on the resources of any cloud providers, and it includes strategies for fault-tolerance. BOT was evaluated using an application that solves a classical combinatorial optimisation problem (binary knapsack). [10] presents a similar framework for developing bag-of-tasks applications, called AzureBOT. However, this framework targets exclusively a single cloud, that is, the Azure cloud. AzureBOT was evaluated using two applications, an Internet data scraper and a master-slave simulator. [11] discusses a data farming platform, called Scalarm, which relies on the master-worker pattern, and presents its extension for running a molecular dynamics simulation on clusters and on clouds. The evaluation of BOT, AzureBOT, and Scalarm focuses on application performance and cost without considering fault-tolerance and elasticity.

3 CASE STUDY: BVDV

This section discusses the use of simulation for studying the spread of bovine viral diarrhea virus. BVDV is an endemic disease in cattle that causes economic losses owing to reproductive disorders, decrease in milk production as well as increases in morbidity and mortality among infected calves [12]. Within the MIHMES project [13], a research group has focused on building a high-performance computer simulation model of BVDV spread among dairy herds at regional scale.

To identify the main drivers of BVDV spread in a region and to compare a large range of regional control strategies, an original stochastic multi-scale model of BVDV spread in discrete time has been proposed based on characteristics of dairy cattle herds in Brittany (western France). The model accounts for the two main transmission pathways between herds: trade movements of animals carrying the virus (mainly uninfected dams carrying a persistently infected foetus, and infected but undetected animals), and virus transmission during the grazing season due to over-the-fence contacts with animals from neighboring herds [14]. In addition, the model accounts for the local within-herd virus spread, using as a building block a previous within-herd model ([12, 15]), adapted to account for heterogeneity among herds in terms of size and management (renewal rate of cows, calving distribution over time, trade movements, etc.).

Practically, the simulation uses a 2 GB database describing cattle life trajectories in Brittany from 2005 to 2013. This database provides information on the demography of dairy animals (e.g., birth, death, culling rates) and trade movements between farms (e.g., holding farms, dates and reasons of entry/exit). The model represents

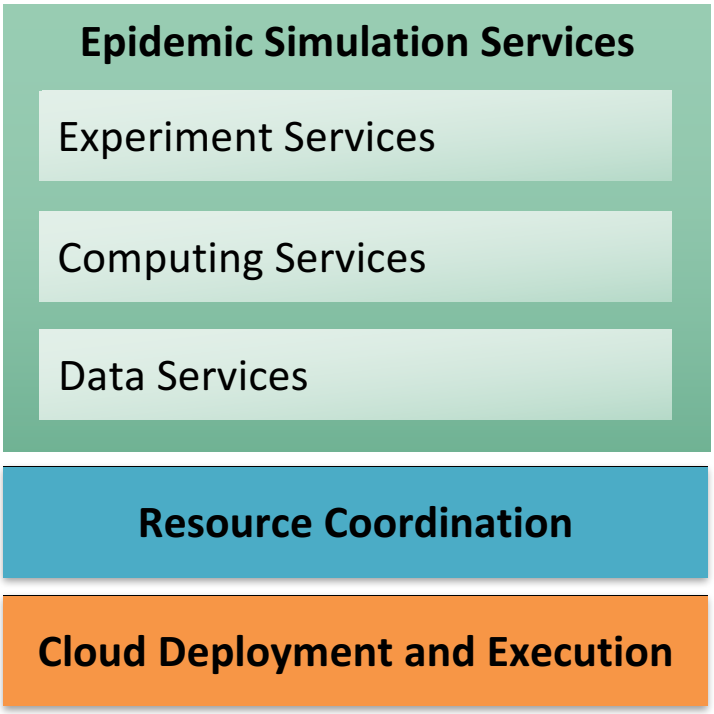


Figure 1: General Architecture of DiFFuSE

the 12,750 dairy cattle herds located in Brittany which raise more than 10 cows, which consisted in more than 2.7 million animals over the 9-year period.

This complex simulation poses the challenges of performance and operational costs, already discussed in Section 1 for generic epidemic simulations. Indeed, performing from 200 to 500 repetitions is required to ensure accurate predictions. This time-consuming and error-prone process might take up to two weeks using a single core assuming that there are no failures from the human or machine sides. Moreover, researchers have to perform simulations for many possible spreading scenarios and carry out sensitivity analyses for many model parameters. Finally, the between-herd model is a fine-grained model containing details on groups of cattle and on animal movements that must correspond to real-world herds and communities. For all these reasons, performing such simulations remains beyond the budget of small and medium laboratories and research groups.

The BVDV case study will be used in demonstrating the DiFFuSE framework and in presenting experimental results for evaluating the framework.

4 DIFFUSE ARCHITECTURE

The DiFFuSE framework for cloud-based epidemic simulations is structured in three layers, as shown in Fig. 1. At the heart of DiFFuSE is the resource coordination layer providing mechanisms to manage resources in a distributed environment. At the top is the epidemic simulation services layer, taking the form of a set of services cooperating to realize the epidemic simulation. At the bottom is a cloud deployment and execution platform responsible for running these services across multiple clouds. The rest of this section provides details on each layer, on failure handling, and on applying the framework in the BVDV case study.

4.1 Epidemic Simulation Services

In DiFFuSE, epidemic simulations are composed of distributed, interacting services, each running as a separate process and communicating over the network. The services share data in a reliable and scalable manner using mechanisms provided by the resource coordination layer, detailed in the next section. The epidemic simulation

services fall into three main categories.

- `Data Services`: retrieve data from databases and provide this data as well as related meta-data to other services.
- `Computing Services`: perform the main computations of the simulation, either sequentially or in parallel, using single-computer parallelisation (e.g., using OpenMP [16]). Computing services may also calculate partial results and generate final results. These services typically obtain data from the data services.
- `Experiment Services`: control the life-cycle of epidemic experiments. They track the progress of the computations, they handle service failures, and they provide configuration data about the experiments to other services.

To support developers in using the framework, DiFFuSE provides three generic C++ classes that serve as templates for developing services of the previously-mentioned categories. All services have a similar structure: they define service routines that are executed when requests are received by the service, and they instantiate a resource manager class that provides access to local and remote resources, as described in detail next.

4.2 Resource Coordination

The resource coordination layer enables services to exchange data in a reliable and scalable way. This layer relies on the concepts of resource, resource manager, resource user, resource provider, and resource monitor. A `resource` is any data that can be named and exchanged between services, such as the experiment configuration, the data configuration, or the experiment data itself. A `resource manager` owns and provides access to local resources and is identified by a URL. Resource managers can be linked in a hierarchy in which a resource manager can provide access to the resources of its parent resource manager. To build this hierarchy, a resource manager specifies the URL of its parent resource manager.

A service can play three roles with regards to a resource, i.e., resource user, resource provider, or resource monitor. A `resource user` sends requests to a resource, demanding the associated data. A `resource provider` receives and replies to such requests, providing access to the resource data. A `resource monitor` tracks the state of a resource (e.g., the resource is inactive, providers are present, providers have failed).

A resource supports one of four management styles determining how data is distributed among providers:

- `Single provider`: the resource has only a single provider offering all the data.
- `Replicated management`: the resource can have multiple providers, each offering the same data. A request to the resource is delivered to any of the providers. This management style can be used to support fault-tolerance; if one provider fails, the requests are handled by the remaining providers. Moreover, it can be used to improve the request response time through balancing the load among providers. A service can obtain the data from an available provider and become itself a provider.
- `Individual management`: the resource can have multiple providers, each offering a different part of the data. A resource user can send requests to each provider to obtain the data.
- `Collective management`: the resource has a fixed set of providers, each offering a different part of the data. A resource user can obtain the data if all providers are available. If a provider fails, a notification is sent that allows reacting to this failure.

Resource coordination in DiFFuSE is implemented using three C++ classes: the `resourceManager` class represents the resource manager, which owns local resources and interacts with its parent resource manager for providing access to remote resources; the `resourceLocalClient` supports interacting with local resources and the `resourceClient` class supports interacting with remote resources.

4.3 Failure Handling

The epidemic simulation is composed of different services, running on multiple servers. The distributed nature of this system introduces many opportunities for failure:

- Deployment-order failures: services may fail when they try to interact with services that are not yet deployed.
- Service failures: services may fail due to, e.g., losing network connection.
- Cleanup failures: services may fail to release allocated resources when the simulation ends.

We next describe how DiFFuSE handles these failures.

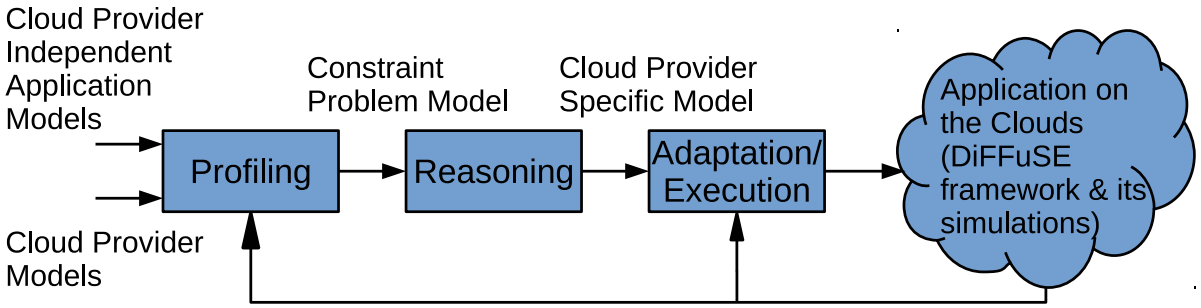


Figure 2: PaaSage components

4.3.1 Deployment-order failures

DiFFuSE allows services to be launched in any orders through using the `nanomsg` message library. With classical TCP sockets, a connection to a service fails if the service is not already running. With `nanomsg`, the library transparently handles connection failures using a retry mechanism, removing any dependencies on deployment order. Connection attempts are timed out to avoid long delays.

4.3.2 Service failures

Each resource manager (except the root manager) is connected to a parent resource manager. The services on each side of such connections check that they periodically receive messages from the other side and take appropriate actions when failures are detected. For instance, if the resource manager detects the failure of a remote service that was a resource provider, the manager updates the resource state and notifies all resource users and monitors.

4.3.3 Cleanup failures

If the `Experiment` service stops or is killed, all other services also stop after a small delay. This guarantees that resources are cleanly released after the end of the simulation.

4.4 Cloud Deployment and Execution

The cloud deployment and execution layer enables hosting the services composing the epidemic simulation on a multi-cloud infrastructure. This task involves selecting appropriate cloud resources (e.g., VMs with specific compute, memory, and storage capacities and prices) from one or more cloud providers (e.g., Amazon EC2, OpenStack-based private clouds), deploying the services on them, and potentially modifying the deployment dynamically, e.g., to handle load variations. Manually performing this task is complex and error-prone, especially if multiple, heterogeneous clouds are used. To manage this complexity, this DiFFuSE layer builds on the PaaSage platform [17], a holistic solution for automatic deployment and execution of cloud applications on multiple clouds. PaaSage consists of three main components:

- **Profiling**: receives descriptions of cloud providers (e.g., VM types, prices) and of applications (e.g., components and interconnections, objectives) and generates a Constraint Problem (CP) model. This model formalizes the optimization problem of how to best deploy the application on available cloud resources. The descriptions are written using CAMEL, the PaaSage cloud application modelling and execution language [17].
- **Reasoning**: solves the CP model using an appropriate solver, such as a mixed integer linear programming solver or a constraint programming solver, and generates a Cloud Provider Specific Model (CPSM), representing the optimal deployment solution.
- **Adaptation/Execution**: deploys the application based on the CPSM through obtaining VMs, installing, and configuring software. Importantly, this component monitors and dynamically adapts the application deployment when run-time changes make this deployment unsatisfactory, e.g., when performance constraints are violated, or cloud provider prices are modified. It can notably trigger elasticity actions (e.g., scale out a component) based on ECA (Event Condition Action) rules defined in CAMEL.

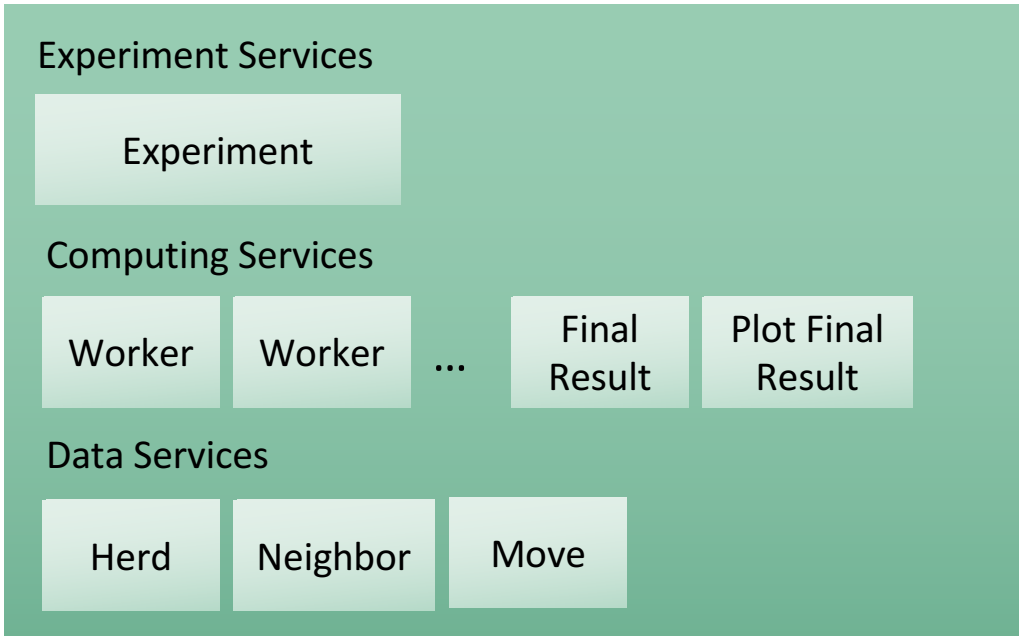


Figure 3: Services of the BVDV spread simulation

In the context of DiFFuSE, PaaSage receives as input CAMEL descriptions of the simulation services and of available cloud providers and generates and dynamically maintains an appropriate deployment involving one or more clouds.

4.5 The BVDV spread simulation

The legacy application for BVDV spread simulation was transformed to a cloud-based application based on the DiFFuSE framework. Specifically, the original sequential code (around 4600 lines of C++ code) was first modified to add single-computer parallelism using OpenMP. The services to be used in the target version were then identified and implemented by extending the generic C++ classes and integrating the existing code. Most of the effort was spent on implementing the data services, which required decomposing the original data model into separate pieces of data (e.g., herds, movements) managed by independent services. Implementing computation services required minor modifications for adapting the original simulation computation code and inserting it into the generic class. The experiment service was largely based on the generic class. In total, we modified around 6.3% of the original code for integration in DiFFuSE. In the final DiFFuSE-based version, the part of the code that is specific to the BVDV spread simulation is around 17.7%.

The services in the DiFFuSE-based application are outlined next (see Fig. 3):

- `Experiment`: controls the life-cycle of a BVDV spread experiment and performs all the tasks of the generic experiment service.
- `Herd`: provides the initial herd description to other services. Different flavors of this service provide herd descriptions generated from the experiment configuration, herd descriptions read from files, and replicated herd descriptions. This replication improves the failure tolerance and responsiveness of the service.
- `Move`: provides movement descriptions to other services. As for herds, different flavors of this service are supported.
- `Neighbor`: provides neighbor descriptions of herds.

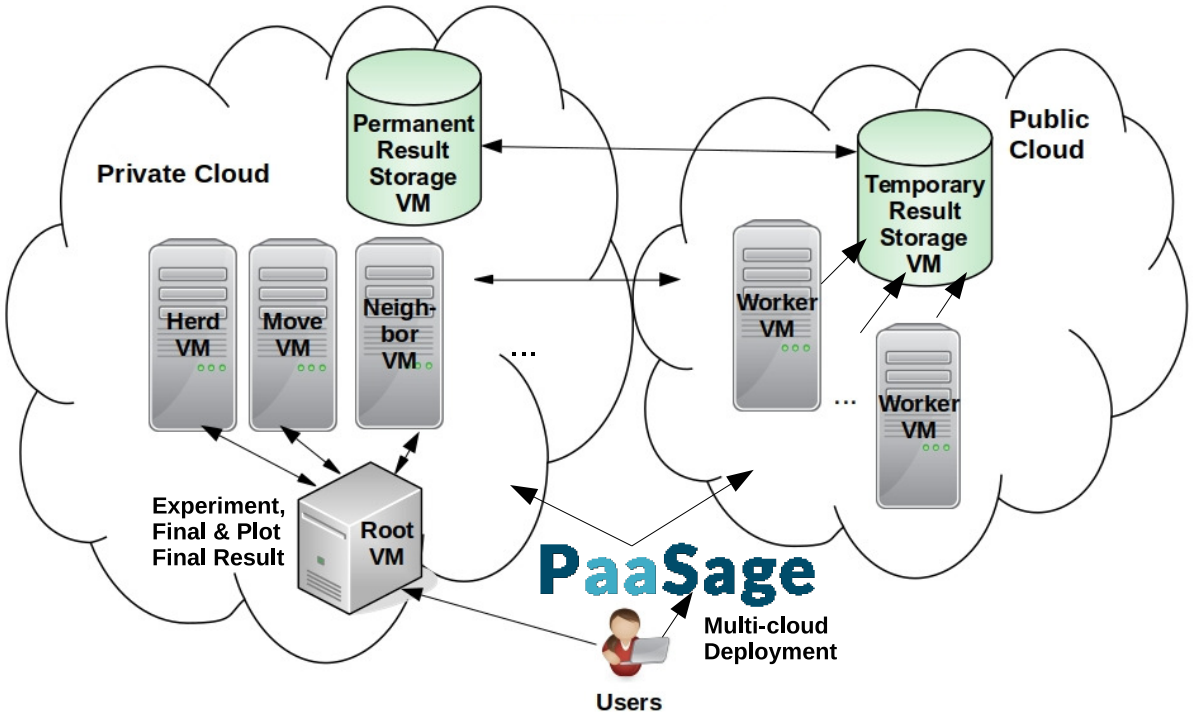


Figure 4: The application for BVDV spread simulation deployed on the cloud

- `Worker`: participates in the simulation's computation. Multiple services of this type can be started on multiple compute servers. Each `Worker` service retrieves the simulation configuration from `Experiment`, herd descriptions from `Herd`, movement descriptions from `Move` and neighbor descriptions from `Neighbor`. Once initial data is received, the worker interacts with `Experiment` to compute partial simulation results and then makes the partial results available for other services.
- `Final Result`: gets partial results from workers, computes final results, and provides the final results on demand.
- `Plot Final Result`: gets final results from workers and uses the `gnuplot` software [18] to generate a graphical view of the results, which can be stored to a file.

The services are deployed through the cloud deployment and execution layer based on PaaSage. Figure 4 shows a possible deployment in a multi-cloud environment.

5 EVALUATION

We conducted a set of experiments to evaluate DiFFuSE in terms of performance, cost effectiveness, failure handling and elasticity in single or multi-cloud environments. The BVDV spread simulation is used throughout the experiments as the reference application. We ran the experiments in three cloud environments: Amazon EC2 (a public cloud), GWDG [19] (a community cloud), and OpenStack on Grid5000 [20] (a private cloud). In Grid5000, we installed OpenStack on nodes of the Parasilo cluster at Inria in Rennes. Each experiment is repeated 5 times and we show the averages.

5.1 Performance and Cost

The legacy BVDV code is originally sequential. To obtain accurate results, one simulation needs to be performed from 200 to 500 times (i.e., 200 to 500 runs). According to experiments done at INRA, performing 500 runs with the sequential legacy code takes around two weeks in the server of the BioEpAR research unit [21]; the server is equipped with 13 cores and 100 GB memory.

5.1.1 Experiment 1

In the first experiment, we evaluate the execution time and corresponding cost when running the DiFFuSE-based simulation in our private OpenStack-based cloud in Grid5000. The cost is calculated as the total price of all used VMs multiplied by the running time. Although the VMs in our private cloud are free, a reference price (in $/hour) is used based on the price of a similar VM type in the Amazon EC2 cloud. Each worker service is packed into one VM with 13 cores and 100 GB memory, which is similar with the configuration of the BioEpAr server. We use the price of the r4.4xlarge VM type in EC2 EU (Ireland) as the reference price.

Fig. 5 shows the time and cost to conduct 20, 40, 100, 200 and 500 runs. The execution time is almost linear in the number of runs. With 500 runs, the execution times are notably 268.36 and 127.76 minutes with 4 and 9 worker services, a great improvement with respect to the 2 weeks in the original BioEpAr experiments. We observe that performance increases when the number of workers rises from 4 to 9, and the effect is greater with increasing number of runs. The cost slightly increases when the number of workers increases from 4 to 9 (less than 2.5$ difference).

5.1.2 Experiment 2

In this experiment, we evaluate the execution time and cost when running the DiFFuSE-based simulation in a public cloud, in particular, Amazon EC2. As each run of the simulation needs at least 35 GBs of memory, we consider two scenarios, each using a different type of memory-optimized EC2 VM: r4.xlarge (4 cores, 30.5 GB RAM + 16 GB swap) and r4.2xlarge (8 cores, 61 GB RAM). In a third scenario, we execute an OpenMP-based BVDV spread simulation, not integrated into DiFFuSE, on a single EC2 VM of r4.2xlarge type. The results are also compared with a fourth, base scenario, which executes the legacy, sequential BVDV code on the BioEpAR server, whose configuration is higher than our EC2 VMs.

Fig. 6 shows the time and cost to conduct 5, 10, 20 and 40 runs with coloured lines representing the 4 scenarios. Only 5 and 10 runs are shown for the base scenario, since execution time and cost increase fast with the number of runs. We see that the DiFFuSE-based BVDV spread simulation with 4 worker services of the r4.2xlarge type outperforms the OpenMP-based simulation with one node. Even with r4.xlarge, a lower configuration, the DiFFuSE-based simulation with 4 worker services still has a better performance at 20 and 40 runs. We also see that the simulation using r4.2xlarge outperforms the one using r4.xlarge. The reason is that r4.xlarge does not have enough RAM (< 35 GB), thus having to consume its swap space in both data loading and transferring phases. This leads to 60% slower time in these phases than in the case of r4.2xlarge, according to our measurements. Regarding cost, the r4.2xlarge scenario costs more than the r4.xlarge scenario except with 10 runs, where the costs are almost equal. The OpenMP-based, single-node scenario has the lowest cost with number of runs less than 40, but this advantage disappears after 40 runs. The base, BioEpAR scenario shows the worst performance and cost in all cases.

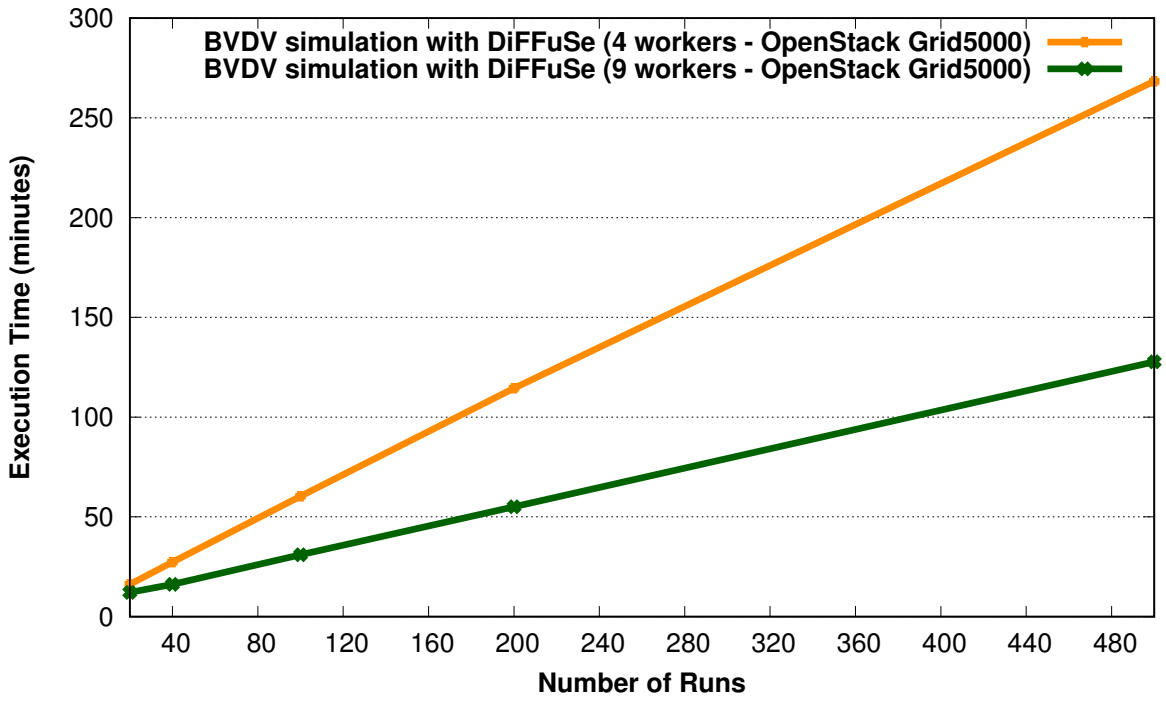
5.1.3 Experiment 3

Assuming that we have enough resources to add more and more workers to the simulation, what will happen with the cost and execution time? To answer this question, we execute the simulation with different numbers of workers (from 1 to 4) with a fixed number of runs (5 and 20). Fig. 7 shows that the cost keeps increasing when the number of workers increases, but the execution time does not keep decreasing. After a specific number of workers, the execution time shows almost no reduction.

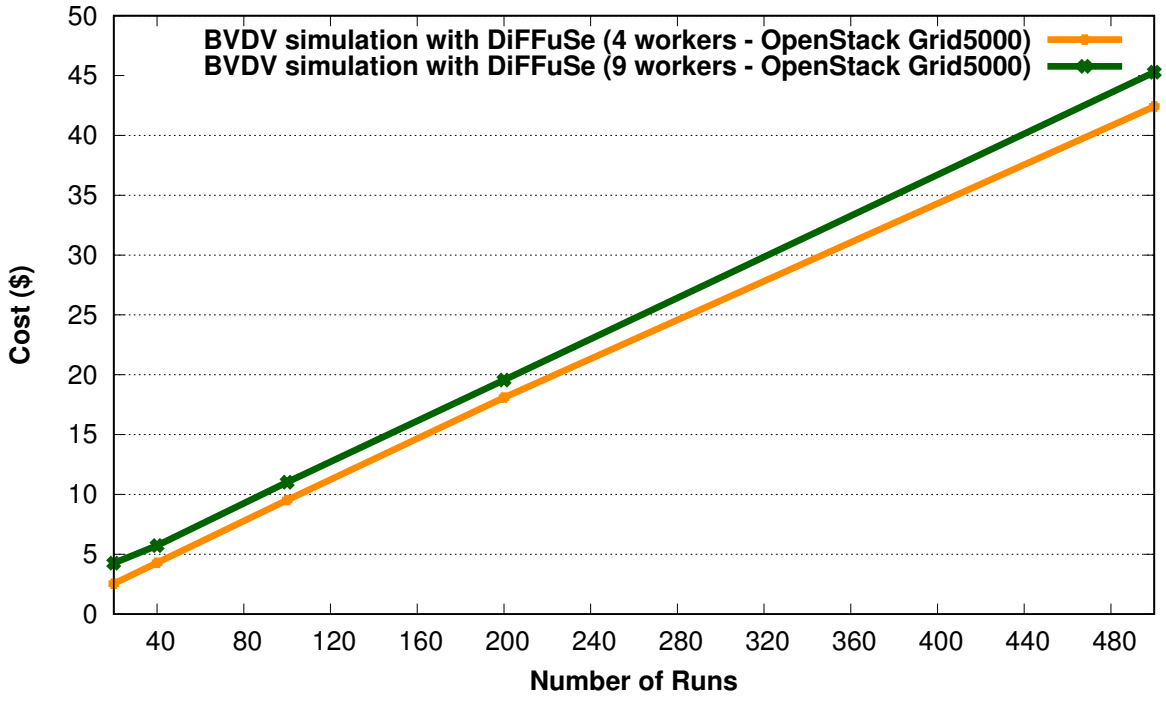
The previous experiments validate that DiFFuSE increases the performance of the BVDV spread simulation while allowing making different cost-performance trade-offs by controlling the number and types of used cloud resources.

5.2 Failure Handling

This experiment evaluates the failure handling capabilities of DiFFuSE. We use the DiFFuSE-based application for BVDV spread simulation with three workers packed in VMs of our OpenStack private cloud (see Section 5.1.1). In one scenario, one of the workers fails early and the others get the burden of redoing the runs until there are zero remaining runs. In the second scenario, the failed worker is revived and takes on further work, if the number of remaining runs is larger than zero. Before the recovery of the failed worker, the other workers recalculate the lost blocks and get other blocks from the experiment service if they are free. We set a maximum two runs for a block that a worker can receive and execute. The simulation is repeated 21 times. Other scheduling algorithms can be implemented to get better performance.

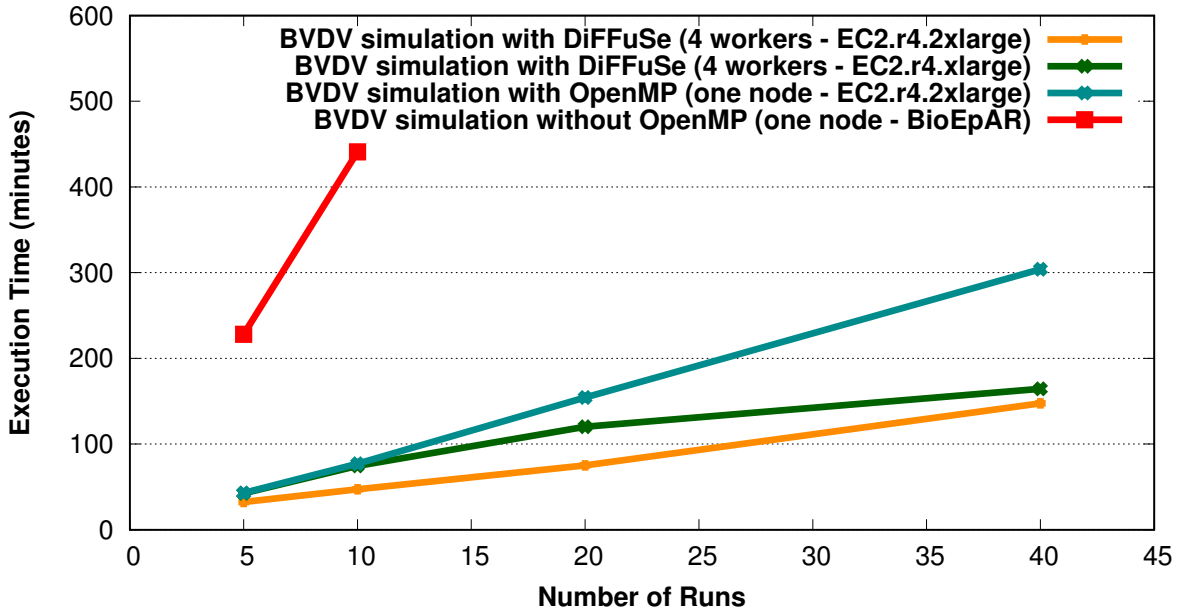


(a)

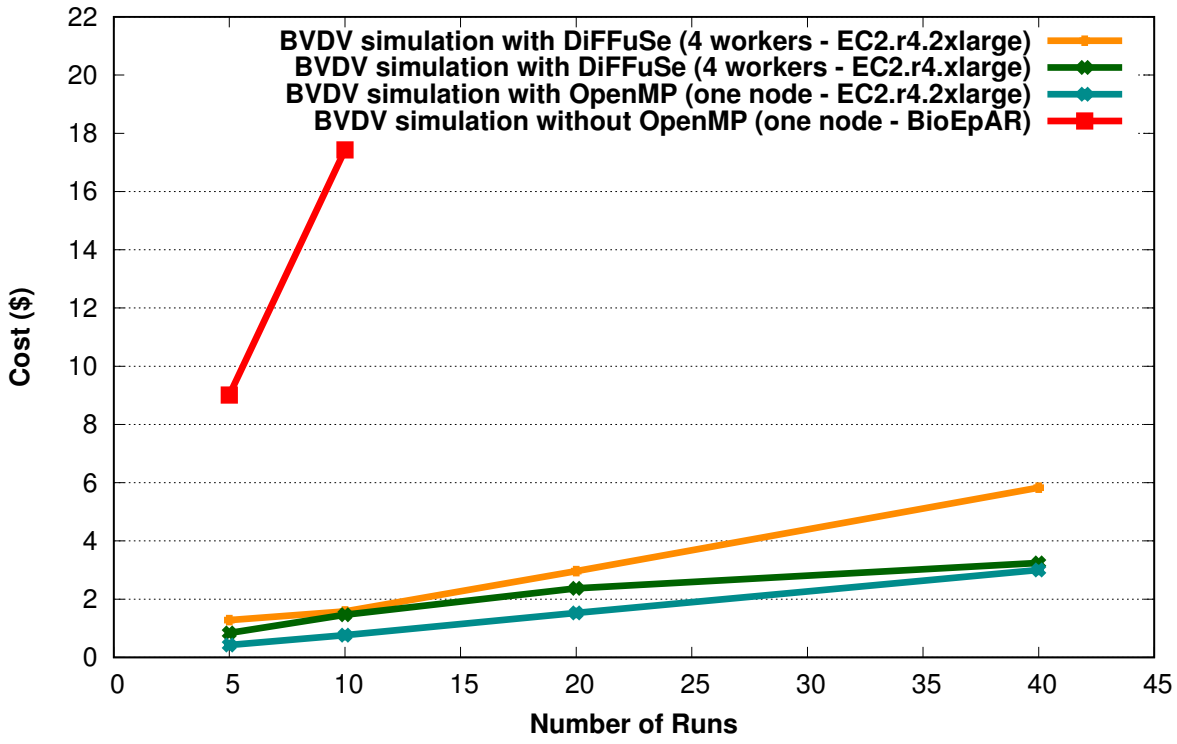


(b)

Figure 5: In Private Cloud: Number of Runs vs a) Execution Time b) Cost

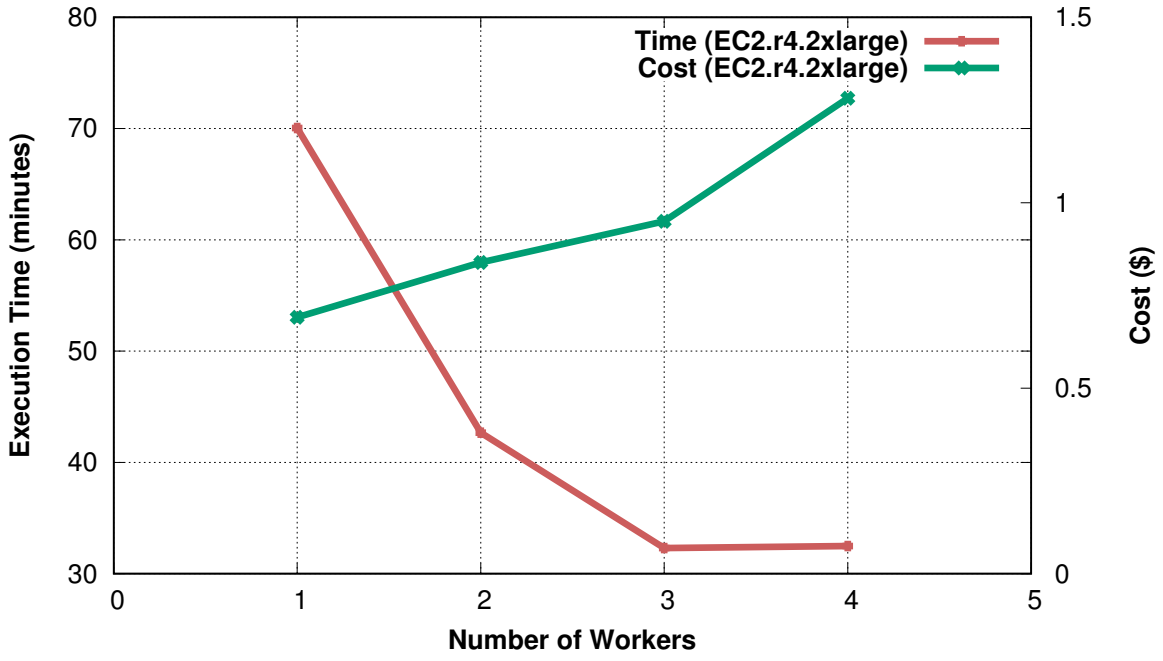


(a)

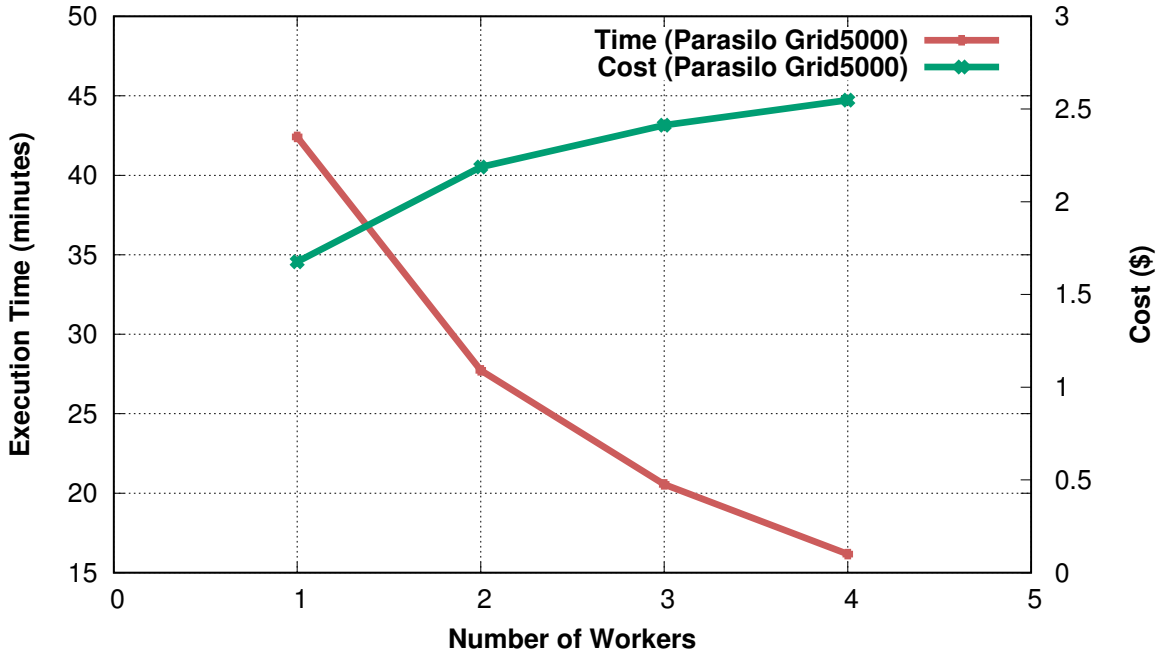


(b)

Figure 6: In Public Cloud: Number of Runs vs a) Execution Time b) Cost



(a)



(b)

Figure 7: Execution Time and Cost to run the BVDV spread simulation: a) 5 runs b) 20 runs

Table 1: THE BVDV SPREAD SIMULATION IN A HETEROGENEOUS MULTI-CLOUD ENVIRONMENT

BVDV spread simulation - Total runs 20							
VM Types	**# Cores**	**# Runs**	**Execution Time**	**Total CPU Time**	**Total Cost**	**R_CPU**	**Cost per Run**
ec2_m4_4xlarge	16	10	$32.62(m)$	$14591.30(s)$	0.651$	46.60%	0.065$
ec2_r4_2xlarge	8	8	$36.10(m)$	$11115.20(s)$	0.435$	64.15%	0.054$
gwdg_m2_xlarge	4	2	$27.93(m)$	$2706.91(s)$	0.217$	40.38%	0.108$

Fig. 8a and 9a demonstrate the number of runs distributed to the three workers over simulation time in the two scenarios. The worker 3 is the failed worker (around the 7th minute) which never recovers in scenario 1 and recovers in scenario 2 (around the 94th minute). The results in Fig. 8b and Fig. 9b show that scenario 2 makes the distribution of runs about 23 minutes faster, which reduces the total execution time. The experiment validates that DiFFuSE enables the simulation to dynamically adapt to service failures and recoveries.

5.3 Multi-Clouds

This experiment evaluates the support provided by DiFFuSE for multi-cloud deployment. We deploy the services of the BVDV spread simulation in 2 VMs (r4.2xlarge and m4.4xlarge) of Amazon EC2 and 1 VM (m2_xlarge - 4 cores 32 GB RAM + 16 GB swap) of GWDG, which is a cloud-serving research community of Göttingen University and the Max Planck Society in Germany. The price of GWDG m2_xlarge VM type is inferred from the EC2 r4.xlarge VM type with an identical configuration. All three workers must collectively execute 20 runs. Table 1 shows the statistics of this experiment.

In this table, total CPU time is the amount of time which multiple cores of a VM were used for processing instructions of the BVDV spread simulation. R_CPU is the exploitation rate of cores in each VM and it is calculated using formula (1).

$$R\_CPU = \frac{TotalCPUTime * 100}{ExecutionTime * NumberofCores} \tag{1}$$

From the table, we see that the experiment service distributes more blocks of runs to VMs with better configuration (e.g., 10 for ec2_m4_4xlarge and only 2 for gwdg_m2_xlarge). Still, the time to finish 10 runs in ec2_m4_4xlarge is not much different than the time to finish 2 runs in gwdg_m2_xlarge. The cost per run of ec2_m4_4xlarge is lower than that of gwdg_m2_xlarge (0.065$ and 0.108$ respectively). While ec2_m4_4xlarge and gwdg_m2_xlarge exploited about 40% to 45% of the potential capacity of their cores, ec2_r4_2xlarge used its cores more efficiently (64.15%). This experiment validates the ability of DiFFuSE to simultaneously exploit diverse VMs from multiple clouds and shows that the choice of VM types has a strong impact on the resulting cost.

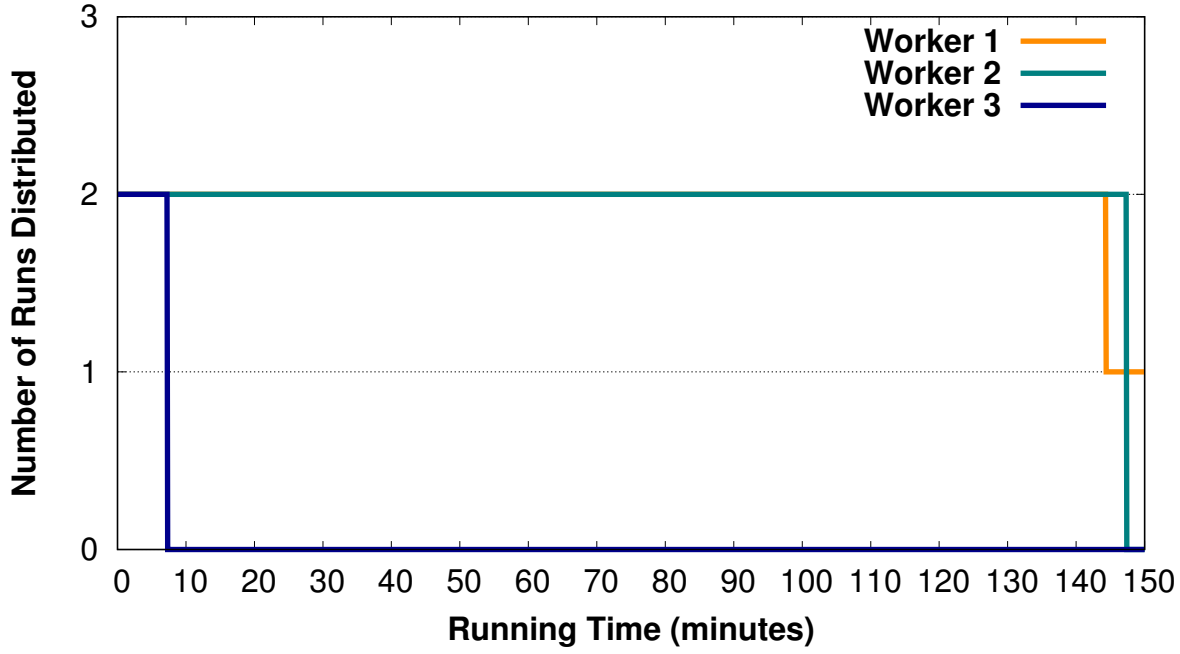
5.4 Elasticity

This experiment evaluates the elasticity support provided by DiFFuSE. We use a scenario in which several dozens of worker services are simultaneously accessing a data service to load initial data, which causes the data service to become a bottleneck. We compare the performance of the simulation with and without the elasticity feature, which enables scaling out the data service.

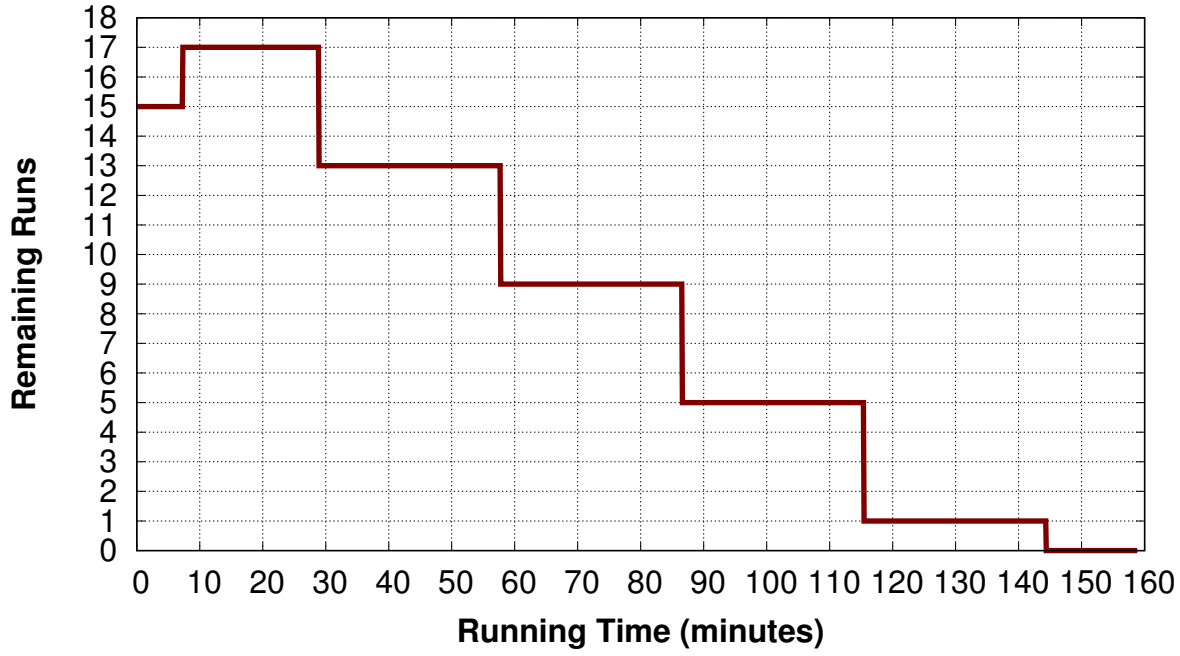
Fig. 10 shows the CPU utilisation (%) of the data service in response to 20 worker services. All services are deployed in EC2 r4.xlarge VMs. The green and orange lines represent the cases with and without elasticity, respectively. The scale-out action is triggered if the CPU utilisation of the data service goes over 80% (the point A), which results in adding one more VM with the data service (green line from the point A to near B). Without elasticity, the single data service is busy with processing requests from the workers (orange line from the point A to C). Importantly, the duration of processing data requests with elasticity is about 30% shorter than without elasticity, showing the usefulness of elasticity support in DiFFuSE.

6 CONCLUSION

This paper addressed the problem of enabling epidemic simulation applications to run in the cloud. Specifically, the paper presented DiFFuSE, a framework that provides design support, reusable code, and tools for building

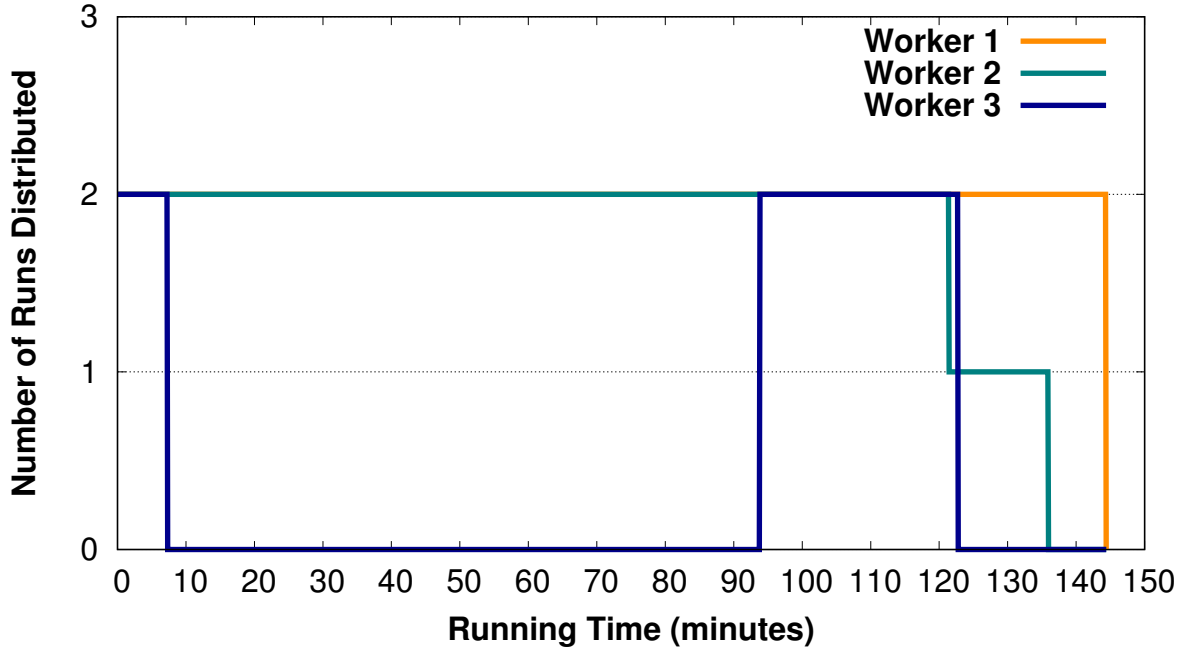


(a)

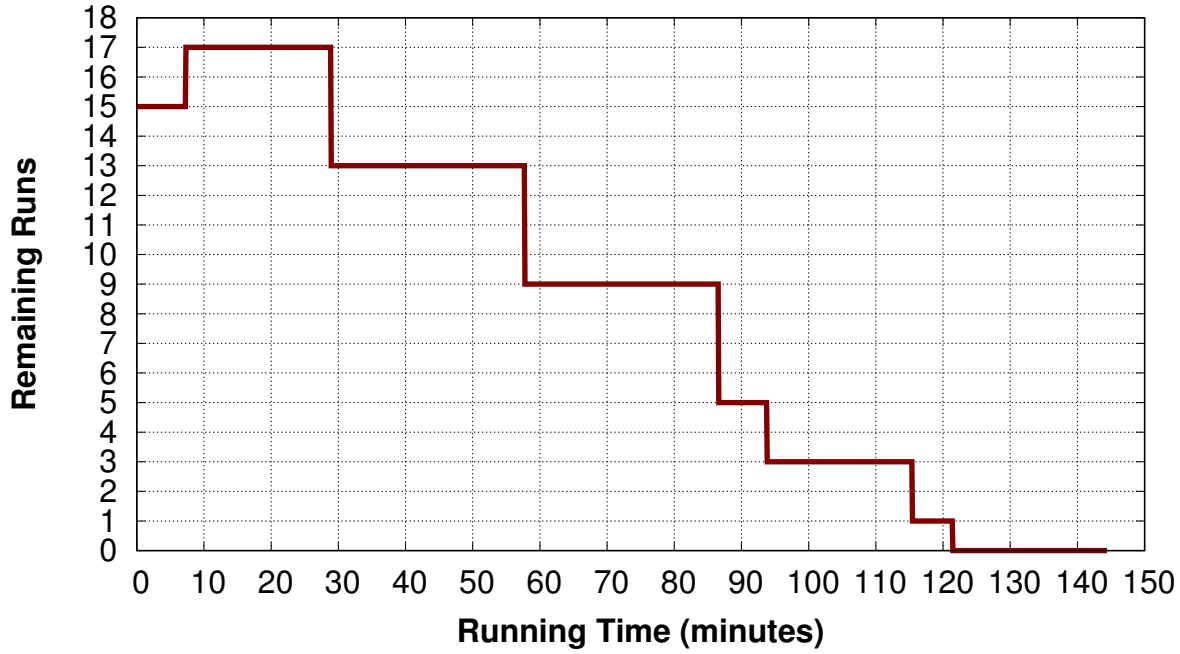


(b)

Figure 8: DiFFuSE with failure handling - Scenario 1



(a)



(b)

Figure 9: DiFFuSE with failure handling - Scenario 2

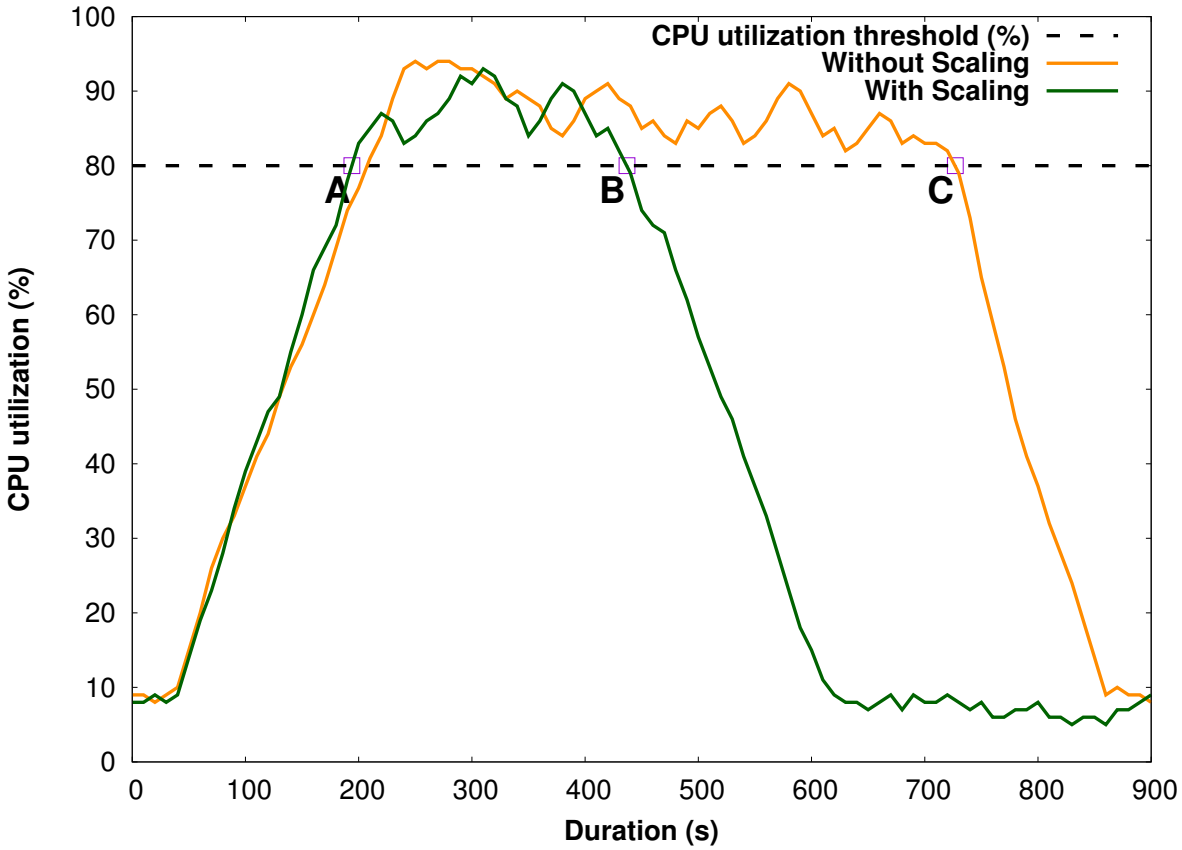


Figure 10: CPU utilisation (%) of the BVDV data service in response to 20 workers - EC2.r4.xlarge

and executing epidemic simulations. The novelty of DiFFuSE is that it helps decomposing epidemic simulation applications into separate services, which are flexibly deployed and managed in multi-cloud environments. The paper also described how DiFFuSE was applied to convert a legacy application (simulation of BVDV spread) to an elastic, cloud-enabled application. A comprehensive set of experiments based on the BVDV case study demonstrated the main benefits of applying DiFFuSE, which are outlined next.

First, the framework exploits variable numbers and types of cloud resources, allowing making different trade-offs between performance and cost. Second, the framework dynamically handles failures and recoveries, minimizing the time and cost of repeating computations. Third, the framework allows using resources from multiple clouds; notably, it allows combining private resources with those of public clouds, enabling, for instance, small institutions, such as local centers for disease control and prevention, to find cost-optimal resource allocations. Finally, the framework allows elastically modifying resource allocations to adapt to dynamic variations in resource demand; this was demonstrated by automatically and independently scaling the data service to avoid a performance bottleneck.

As future work, we intend to apply DiFFuSE to further simulation applications, which will provide useful feedback for improving the framework and allow a more complete evaluation.

Acknowledgments

This work was carried out with the financial support of the French Research Agency (ANR), Program Investments for the Future, project ANR-10-BINF-07 (MIHMES). The authors would like to thank Yvon Jégou for developing the original framework. We would also like to thankfully acknowledge the support of the PaaSage (FP7-317715) EU project, the European fund for the regional development of Pays de la Loire (FEDER), and the European fund for inventive researchers (Agreenskills plus).

References

[1] B. L. Dutta, P. Ezanno, and E. Vergu, "Characteristics of the spatio-temporal network of cattle movements in France over a 5-year period," *Prev. Vet. Med.*, vol. 117, no. 1, pp. 79–94, Nov 2014.

[2] S. Widgren, S. Engblom, P. Bauer, J. Frössling, U. Emanuelson, and A. Lindberg, "Data-driven network modelling of disease transmission using complete population movement data: spread of vtec o157 in swedish cattle," *Veterinary Research*, vol. 47, no. 1, p. 81, Aug 2016.

[3] G. Beaunee, E. Vergu, and P. Ezanno, "Modelling of paratuberculosis spread between dairy cattle farms at a regional scale," *Vet. Res.*, vol. 46, p. 111, Sep 2015.

[4] H. Eriksson, T. Timpka, A. Spreco, O. Dahlstrom, M. Stromgren, and E. Holm, "Dynamic multicore processing for pandemic influenza simulation," in *AMIA Annual Symposium Proceedings*, 2016, pp. 534–540.

[5] W. W. Yim, S. Chien, Y. Kusumoto, S. Date, and J. Haga, "Grid heterogeneity in in-silico experiments: an exploration of drug screening using DOCK on cloud environments," *Stud Health Technol Inform*, vol. 159, pp. 181–190, 2010.

[6] H. Eriksson, M. Raciti, M. Basile, A. Cunsolo, A. Froberg, O. Leifler, J. Ekberg, and T. Timpka, "A cloud-based simulation architecture for pandemic influenza simulation," in *AMIA Annual Symposium Proceedings*, Oct. 2011, pp. 364–73.

[7] P. Sukcharoen, S. Pumma, O. Mongkolsermporn, T. Achalakul, and X. Li, "Design and analysis of a cloud-based epidemic simulation framework," in *ECTI-CON 2013*, May 2012, pp. 1–4.

[8] R. C. Price, W. Pettey, T. Freeman, K. Keahey, M. Leecaster, M. Samore, J. Tobias, and J. C. Facelli, "Satscan on a cloud: On-demand large scale spatial analysis of epidemics," in *Online Journal of Public Health Informatics; 2(1): ojphi.v2i1.2910*, 2010.

[9] A. D. Benedictis, M. Rak, M. Turtur, and U. Villano, "A framework for cloud-aware development of bag-of-tasks scientific applications," *Int. J. Grid Util. Comput.*, vol. 7, no. 2, pp. 130–140, Jan. 2016.

[10] D. Agarwal and S. K. Prasad, "Azurebot: A framework for bag-of-tasks applications on the azure cloud platform," in *2013 IEEE IPDPSW*, May 2013, pp. 2139–2146.

[11] D. Król, M. Orzechowski, J. Kitowski, C. Niethammer, A. Sulisto, and A. Wafai, "A cloud-based data farming platform for molecular dynamics simulations," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, Dec 2014, pp. 579–584.

[12] P. Ezanno, C. Fourichon, A. F. Viet, and H. Seegers, "Sensitivity analysis to identify key-parameters in modelling the spread of bovine viral diarrhoea virus in a dairy herd," *Prev. Vet. Med.*, vol. 80, no. 1, pp. 49–64, Jun 2007.

[13] "Mihmes," Website http://www6.inra.fr/mihmes.

[14] L. Qi, E. Vergu, B. L. Dutta, and P. Ezanno, "Modeling bovine viral diarrhea virus (bvdv) spread between dairy cattle farms at a regional scale: relative contribution of two between-herd transmission pathways," Mar 2017, presented at Society for Veterinary Epidemiology and Preventive Medicine (SVEPM), Inverness, GBR.

[15] P. Ezanno, C. Fourichon, and H. Seegers, "Influence of herd structure and type of virus introduction on the spread of bovine viral diarrhoea virus (BVDV) within a dairy herd," *Vet. Res.*, vol. 39, no. 5, p. 39, 2008.

[16] "Openmp," Website http://www.openmp.org/.

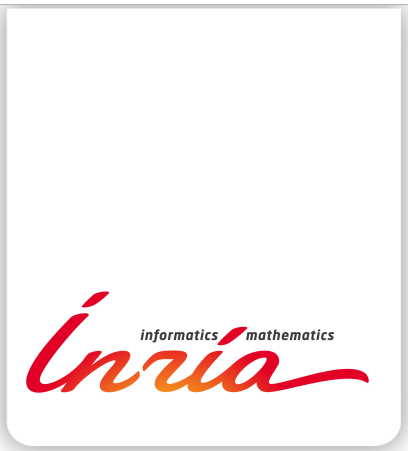
[17] "Paasage," Website https://www.paasage.eu.

[18] "Gnuplot," Website http://www.gnuplot.info.

[19] "Gwdg," Website https://www.gwdg.de.

[20] "Grid5000," Website https://www.grid5000.fr.

[21] "Bioepar," Website https://www6.angers-nantes.inra.fr/bioepar.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399