Computers and Electrical Engineering 77 (2019) 314–324

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Cyber-physical application monitoring across multiple clouds[☆],[☆☆]

Ayman Noor[a,b,*], Karan Mitra[c], Ellis Solaiman[a], Arthur Souza[d], Devki Nandan Jha[a], Umit Demirbaga[a,e], Prem Prakash Jayaraman[f], Nelio Cacho[d], Rajiv Ranjan[a]

[a] *Newcastle University, Newcastle upon Tyne, UK*
[b] *Taibah University, Madinah, Saudi Arabia*
[c] *Luleå University of Technology, Skellefteå, Sweden*
[d] *Federal University of Rio Grande do Norte, Natal, Brazil*
[e] *Bartin University, Bartin, Turkey*
[f] *Swinburne University of Technology, Melbourne, Australia*

ARTICLE INFO

Article history:
Received 30 November 2018
Revised 6 June 2019
Accepted 13 June 2019

Keywords:
Cyber-physical system
Monitoring
Linear road benchmark
QoS
Virtualization
Cloud computing

ABSTRACT

Cyber-physical systems (CPS) integrate cyber-infrastructure comprising computers and networks with physical processes. The cyber components monitor, control, and coordinate the physical processes typically via actuators. As CPS are characterized by reliability, availability, and performance, they are expected to have a tremendous impact not only on industrial systems but also in our daily lives. We have started to witness the emergence of cloud-based CPS. However, cloud systems are prone to stochastic conditions that may lead to quality of service degradation. In this paper, we propose M2CPA - a novel framework for multi-virtualization, and multi-cloud monitoring in cloud-based cyber-physical systems. M2CPA monitors the performance of application components running inside multiple virtualization platforms deployed on multiple clouds. M2CPA is validated through extensive experimental analysis using a real testbed comprising multiple public clouds and multi-virtualization technologies.

1. Introduction

CPS is an interdisciplinary approach for combining communication devices, computation, and actuation for performing time-constrained actions in a predictive and adaptive manner [2,3]. This is done using a feedback loop within the physical system, which enables the embedded and network systems to monitor and control the physical processes. In this way the design of a previous model can be modified using feedback from the physical system. This also makes the system more robust, reliable and free from any past errors. According to the National Institute of Information and Communication

☆ This journal paper is a significantly extended version of conference paper published by IEEE Cloud 2019 conference [1].
☆☆ This paper is for CAEE special section SI-csc. Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. Xiaokang Zhou.
* Corresponding author.
E-mail address: aymannoor84@gmail.com (A. Noor).

https://doi.org/10.1016/j.compeleceng.2019.06.007
0045-7906/© 2019 Elsevier Ltd. All rights reserved.

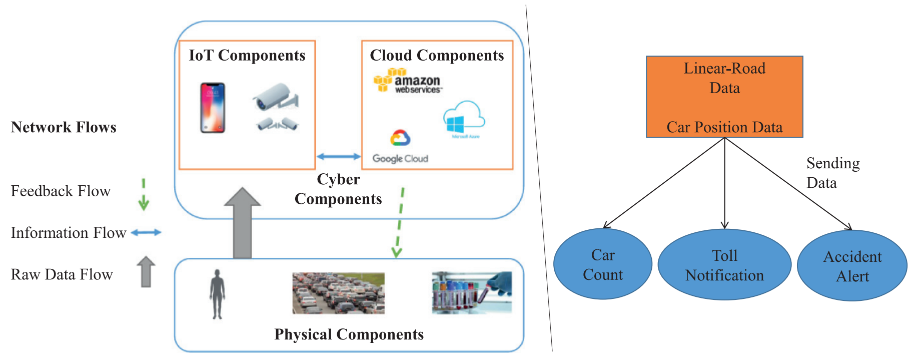

Fig. 1. Cyber-physical system and an example of stream data management for highway monitoring system.

Technology (NIST) [4], cyber-physical cloud computing is *"a system environment that can rapidly build, modify and provision cyber-physical systems composed of a set of cloud computing based sensors, processing, control, and data services"*.

CPS consists of three main elements: cyber, physical, and network components. Each of these components consists of a few other components. For example, the cyber component consists of two components: cloud and IoT devices where the IoT devices work as a bridge between physical and cyber components. The network component is used for interlinking the cyber and physical components and transferring and controlling data as shown in Fig. 1. In order to develop a robust architecture for a CPS solution, data needs to be collected from various physical sources (for example traffic, education, and healthcare systems [5]) using IoT devices (e.g. sensor, mobile, and a camera). Every day larger applications with more devices are being connected with CPS, which means that a larger variety of physical conditions need to be considered, and this requires larger volumes of data to be extracted using IoT devices, and filtered and processed using cloud data centres (cloud). Therefore the main components of a CPS can be summarised as follows:

1. Physical Component: This component does not have any computation or communication capability; it only includes biochemical processes, mechanical processes, or humans. Physical components collect and provide data, which is required to be processed in real time for controlling various activities. Such data is usually highly concurrent and dynamic.
2. Cyber Component: is used for collecting, processing, reporting and controlling all the physical components within CPS. As it is challenging to manage the concurrent and dynamic data from the physical component of CPS, the cyber component is divided into two sub-systems. These are cloud data centers, and IoT devices [5].
3. Network Component: is responsible for communication between the physical and cyber components or among the cyber components. The raw data is captured from components such as IoT devices and passed to the cloud. Also, cloud devices send control and feedback to the IoT devices using network components. Main factors that affect network communications are bandwidth, topology, latency, and congestion [6,7].

1.1. Research context

Fig. 1 describes a conceptual implementation of highway traffic monitoring services using a cyber-physical system. The sensed data of highway traffic (for example the position of the cars) is sent as a stream of events that is physically separated and used for problems such as traffic monitoring and management. This requires the processing of huge volumes of data with high efficiency using the capabilities of multi-cloud environments [3,8,9].

To effectively explore data processing in a multi-cloud environment, three services for highway traffic are considered. These are: (i) Toll Collection Notification, (ii) Accident Alerts, and (iii) Car Count (a detailed discussion is given in Section 4). The system will manage its resources in terms of sensor data and other saved data available in the cloud and provide the requested information to the driver. For example, the highway traffic system will send an alert to drivers on their navigation systems to inform them to take appropriate routes (push mode). Also the driver can request information about traffic routes, and then make informed decisions based on that information (pull mode).

The performance of a cyber-physical application in cloud systems may vary considerably due to factors such as application type, interference effect (caused by other applications running in the same or different containers), resource failure and congestion. Quality of Service (QoS) denotes the levels of service offered by the cloud provider in terms of service features depending on the user's/application's requirements [10]. QoS is generally defined in terms of application specific features such as availability, pricing, capacity, throughput, latency, and reliability or user dependent features such as certification, reputation, and user experience rating. QoS is essential for both the user who expects the cloud provider to deliver the published services, and the provider who needs to find a balance between the offered service and functional cost. Agreement between the user and the provider on the quality of service offered leads to a Service Level Agreement (SLA) [11]. SLA creates transparency between user and cloud provider by defining a common ground, which is agreed by both user and cloud provider. Appropriate penalties are normally associated with the SLA, which are applied in case of SLA violations. Therefore, it is imperative to monitor the QoS provided by the cloud provider to check whether the SLA is satisfied or not. Monitoring is required for different purposes such as resource provisioning [12], scheduling [13–15], security [16], and re-encryption [17,18]. To detect any performance anomaly or to ensure that SLA requirements are achieved, continuous monitoring is essential [19].

In virtualized environments, an application may be distributed over multiple containers/VMs, each running some services communicating over REST-based APIs [20]. Monitoring is required at both individual container/VM level or at application level to guarantee that the QoS requirements of the application are satisfied. There are some lightweight endpoints available that can easily be plugged in to perform the monitoring operations for a single environment application. However, for complex containerized applications, it is challenging to have a single monitoring end-point, because each container may be hosted on different environments that do not support a common monitoring endpoint.

1.2. Research contributions

Currently, there are multiple monitoring frameworks e.g. Docker stat, CAdvisor, DataDog, Amazon CloudWatch, CLAMS [21], available to monitor the applications running in the cloud. However, most of the frameworks are either cloud provider specific e.g. Microsoft Azure Fabric Controller, or virtualization architecture specific e.g. CAdvisor. These monitoring tools are not able to satisfy the complex dependent requirements of CPS that can provide holistic monitoring across multi-cloud scenarios supporting different types of virtualization. Monitoring the performance of services in such a complex environment is very challenging for the following reasons:

- The deployment environment for cyber-physical applications in multi-cloud environments is very complex as there are numerous components running in heterogeneous environments (VM/container) and communicating frequently with each other using REST-based/REST-less APIs. In some cases, multiple components can also be executed inside a container/VM making any failure or anomaly detection very complicated. It is necessary to monitor the performance variation of all the service components to detect any reason for failure.
- Considering the virtualization environment, deployment of cyber-physical applications in containers is very different from that in VM. Containers are defined in terms of namespace and cgroups that share the same host machine whereas each VM is isolated with its own operating system. Also, the resource limitation in containers can be hard or soft as compared to VM which is always strict (hard). A soft limit allows containers to extend beyond their allocated resource limit creating higher chances of interference [22]. Monitoring the performance of cyber-physical applications in such cross VM-container scenarios is very important to ensure that services are executing in a desirable way.
- Modern applications can be distributed across multiple cloud environments including bare metal, public or private cloud depending on several features such as cyber-physical application component requirements, deployment locations, security concerns, cost, etc. Different cloud providers have their own way of handling deployment and management of cyber-physical application components. Due to the heterogeneity of cloud providers, it is complex to have holistic management of application components.

Based on the aforementioned challenges, this paper addresses the following research questions:

- How to monitor the performance of distributed software components of cyber-physical applications running on heterogeneous virtualization platforms within the same or different cloud service providers?
- How to aggregate QoS measures of cyber-physical applications running in multiple cloud environments to give a holistic view of performance?

To answer these questions, this paper makes following new contributions:

- It introduces a novel framework: Multi-virtualization, Multi-cloud Monitoring in Cyber-Physical Applications (M2CPA) that provides a holistic approach to monitor the performance of CPS applications composed into multiple applications deployed/running in a multi-cloud and heterogeneous environment (e.g. using different virtualization technologies).
- It validates the proposed monitoring framework M2CPA, via a proof of concept implementation that monitors cyber-physical application performance running across different cloud service providers using different virtualization means. Experimental analysis verifies the efficacy of our proposed monitoring framework.

The rest of this paper is organized as follows. Section 2 discusses recent related work. The M2CPA framework design is presented in Section 3. Section 4 presents the proof of concept implementation of M2CPA and Section 5 discusses the outcomes of experimental evaluation. The paper concludes by giving some future work suggestions in Section 6.

2. Related work

There are already industry monitoring tools whether in containers [Docker, CAdvisor, Datadog] or in cloud [CloudWatch, Microsoft Azure Fabric]; and academic monitoring tools whether in VMs [14,21] or even in containers [23,24].

Table 1
Comparison of related work.

Parameter(monitoring)	Related work							M2CPA
	Docker	CAdvisor	Datadog	CloudWatch	CLAMS	Microsoft Azure Fabric	PyMon	
Virtual Machine (VM)	✗	✗	✗	√	√	√	✗	√
Container	√	√	√	√	✗	√	√	√
Multiple Cloud	√	√	√	✗	√	✗	√	√
Cyber-Physical Systems	√	√	√	√	✗	√	✗	√

Docker[1] provides an inbuilt monitoring tool, Docker stats, to examine the resource usage metrics of running containers. The various metrics provided by Docker stats are CPU and memory usage, and actual free memory for each container. However, it does not inspect the performance of individual applications running inside a container. Our proposed framework, M2CPA, improves on this significantly by monitoring the performance of each application running inside a container. Along with this, M2CPA also gathers the monitoring information from containers running in heterogeneous cloud environments (e.g. Amazon, Azure, Openstack, etc.). By aggregating the data collected from multiple containers running across multi-cloud environments, one can perform different types of performance comparisons to assess the performance in containers.

CAdvisor[2] is an open source monitoring framework that displays monitoring performance and resource usage in real time. It provides CPU usage, memory usage, network and throughput information of the running containers. One can access the monitoring information only for two minutes duration, as there is no associated storage mechanism that can retain the data for a longer interval. In contrast M2CPA monitors the performance of individual applications that run inside the container/VM and also stores monitoring data in a database shared by both container and VM.

Datadog[3] is a monitoring service that gathers metrics such as CPU utilization, memory, and I/O for all containers. It is an agent-based system that sends data only to the Datadog cloud, making the monitoring job completely dependent on Datadog's cloud. On the other hand M2CPA has the ability to store data in any cloud service provider.

CloudWatch[4] is a commercial cloud monitoring tool that tracks CPU, memory usage, and network but cannot monitor application-level QoS metrics. In addition, it is not platform independent (i.e., it works only for Amazon platform and not for Azure). Similarly, Microsoft Azure Fabric[5] Controller is limited to work only on the Azure platform. M2CPA, on the other hand, has the ability to monitor applications in heterogeneous cloud environments.

In [21] the authors present CLAMS, an application monitoring framework for multi-cloud platforms. Moreover, their monitoring framework considers different QoS parameters for web-applications running inside a VM. The model retrieves the QoS performance for different cloud layers. However, the model does not monitor the performance of containers. In addition, the model is constrained to only web applications. This is different to our framework, which monitors cyber-physical applications that run inside containers and VMs.

In [23] the authors present a framework called PyMon that collects resources like CPU utilization, memory utilization, and network by using Docker container management API. In contrast to [23], our study uses standalone libraries to monitor applications inside the virtualization environment (e.g. containers) and hence can work in heterogeneous environments (e.g. from VM to container). The work published in [24] presents a study between the uses of Virtual Machines and Docker containers comparing the QoS parameters evaluation. They only use Docker containers for their experiments. The authors use the Docker container process to monitor the CPU utilization but they do not validate any application specific parameters of the containers that are being executed.

In [14], the authors present an architecture that collects, analyses and presents the physiological data. Also, it captures data from numerous sensors for further transformation and analysis. This paper is mainly concerned with monitoring particular parameters for performing scheduling in only the cloud environment. In contrast, our framework monitors the performance of an application in a holistic cyber-physical system.

Existing monitoring solutions discussed above do not have the ability to monitor the performance of cyber-physical applications running inside multi-virtualization heterogeneous cloud environments (container/VM). A comparison of different related works with our proposed M2CPA is presented in Table 1.

Our proposed work differs from the aforementioned solutions in that it can be used to holistically monitor the performance of cyber-physical applications components running inside containers and VMs in multiple cloud environments.

3. M2CPA monitoring framework design

This framework consists of two main components namely a monitoring manager and a monitoring agent. Monitoring agents (represented as $\tilde{A}$) are placed inside containers/VMs that track the performance of underlying applications. A mon-

[1] https://www.docker.com.
[2] https://github.com/google/cadvisor.
[3] https://www.datadoghq.com.
[4] https://aws.amazon.com/cloud-watch.
[5] https://azure.microsoft.com.

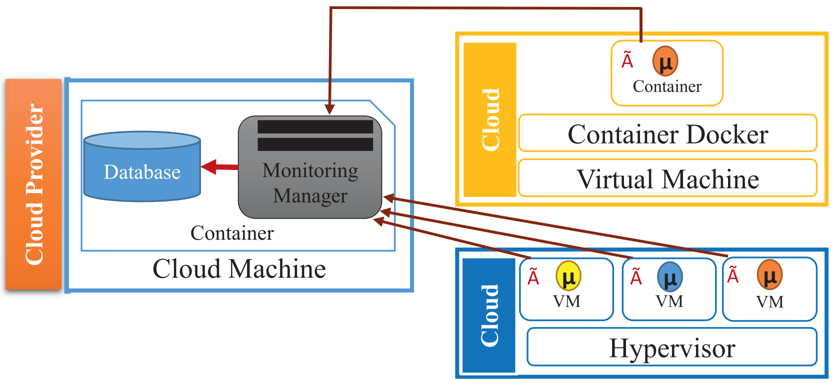

Fig. 2. Overview of M2CPA framework.

itoring agent collects the system-level statistics for each service and sends the information to the manager. The manager deployed in a distant multi-virtualization, collects the information from different monitoring agents and stores this data in a database for further performance analysis and prediction. The configuration of multi-virtualization (containers/VMs) can be either homogeneous or heterogeneous each of which is provisioned on different cloud providers. Each container/VM may execute one or more services of the same or different types. Fig. 2 presents a high-level view of the M2CPA framework.

A detailed discussion on the design of the monitoring agent and monitoring manager is given below.

3.1. Monitoring agent

The monitoring agent is a software component that collects the information from applications running inside (containers/VMs). It has the ability to work in different cloud platforms. Agents will wait for requests coming from the manager to push monitoring information to the manager. M2CPA uses HTTP request for communicating system information between agents and managers. The agents are implemented using the SIGAR (https://github.com/hyperic/sigar/) and RESTLet (https://restlet.com/) libraries that enable them to run on any cloud providers. SIGAR is a multiplatform library (Unix, Windows, Solaris, FreeBSD, Mac OS, etc.) written in Java that provides an API for accessing operating system information while the RESTLet is a Java library that makes it easy to develop HTTP REST APIs.

The M2CPA framework uses SIGAR to obtain the defined system parameters, namely CPU usage, memory usage, free memory, network usage, etc. RESTLet is used in the development of the services of the monitoring agents that would be accessed by the manager to obtain monitoring data.

The monitoring agent is packaged into a jar file and configured to run during the multi-virtualization (container/VM) boot process. First, agent registration information must be sent to the manger using HTTP PUT request. Second, the agent will send data periodically to the manager using HTTP POST request. Finally, agent configuration will be sent to the manager by using HTTP GET request that can update agent configuration parameters. The agent utilizes functionalities provided by SIGAR to retrieve the application metrics and other custom built APIs. SIGAR helps in getting the information parameters for the specific application. Using these functionalities, the agent monitors the specified features for each application ID. The agent will start to retrieve the information parameters for this application such as CPU utilization, memory utilization, and so on. The manager utilizes a pull technique that retrieves the information parameters from all the distributed agents and stores them in an SQL database.

3.2. Monitoring manager

The monitoring manager is a software component that receives monitoring information from agents deployed inside (containers/VMs) scattered in the heterogeneous cloud environment, and provides an API for accessing data saved by other services or other applications. Communication between manager and agents is based on pull-based or push-based mechanisms. The manager makes use of the RESTLet library in building the clients accessing the services of the agents. For each registered monitoring agent, the manager starts a thread that coordinates a RESTLet client for access to agent data. Each time the data of a monitor agent is received the manager stores the results in a MySQL database for further access by the graphical management tool.

The sending of information by the monitoring agent to the monitoring manager occurs as a sequence of steps: First, agent sends a registration request to the manager, and the manager receives the request and registers the agent, an access key and an endpoint are sent with the data returning to the agent. Second, the manager executor (uses Push mechanism) is enabled to receive the data sent by the agent using their IP address. Lastly, the agent periodically queries the manager for its configuration (Change Configuration). Dynamic configuration enables real-time agent management.

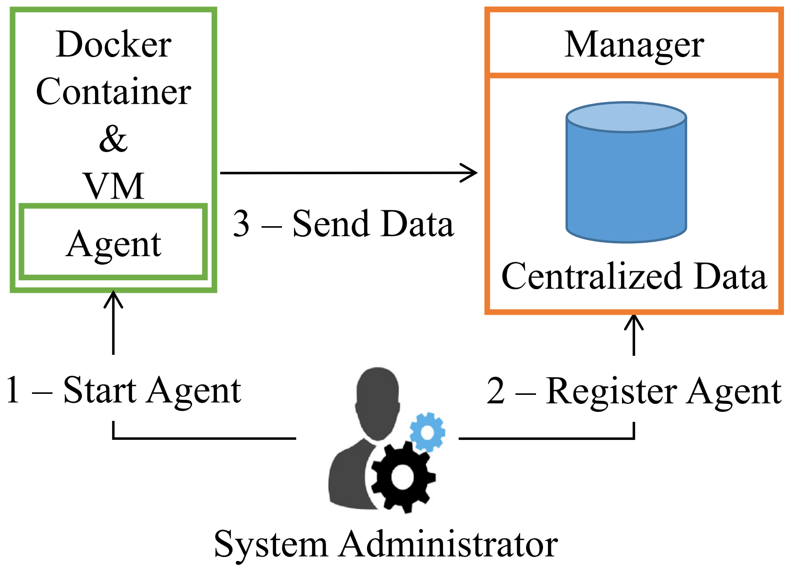

Fig. 3. M2CPA data acquisition model.

The complete monitoring application is represented in the form of a data acquisition model as given in Fig. 3. It consists of three steps. Initially, the system administrator starts the monitoring agent (Step 1). Subsequently, the administrator registers the agent (An HTTP PUT request registers the agent's IP) to the manager (Step 2). The agent continuously monitors the system (applications, containers, or VMs). Finally, all the monitoring agents send the monitored information periodically to the manager using the HTTP POST request. (Step 3). The manager stores the received data in a shared database and also processes any query (if received) related to the performance of the applications.

4. M2CPA implementation

Our M2CPA monitoring framework is implemented in Java and works for both containers and VMs running on any host operating system (Linux, Windows or Mac OS) running on any cloud platform.

To validate the M2CPA framework, we built a highway monitoring system for automated toll collection notification, accident alerting, and car counting using a stream data management system. The choice to use these three applications is justified by the need to evaluate the effectiveness of M2CPA in a variety of scenarios running on a distributed and multi-cloud environment and with different virtualisation techniques.

The highway monitoring application was built on the basis of work published in [25] which presented the Linear road benchmark for evaluating stream data management systems. Through a simulator called MIcroscopic Traffic SIMulation Laboratory (MITSIMLab) it is possible to construct traffic descriptor files of vehicles that travel on a high-road (http://www.cs.brandeis.edu/~linearroad/mitsiminstall.html). The generated data is used as tuples to be sent to the flow processing system. In [25] the authors define some queries that use the data generated in the context of a motorway monitoring application. Following the authors' proposal, we run MITSIMLab and generate a file corresponding to 3 h of vehicular traffic. We programmed three historical data queries: one for toll billing notification; another to detect accidents; and finally to count the number of cars in each track and segment of the highway in real time. Queries were implemented using Esper (http://www.espertech.com/esper/). Esper is a language and an execution machine for processing events and focusses on dealing with high-frequency time-based event data as presented in Fig. 4.

Queries were built to cover constraints and conditions imposed by the Linear road benchmark. Therefore the tuple was used to simulate the position of a car at a certain instant of time. This data was encapsulated in an event composed of the attributes present, and represented the flow of information coming from the positions of the vehicles reported through sensors as shown in Table 2.

The data sent is: TIME, VID, SPD, XWAY, LANE, DIR, SEG. TIME represents the instant of time in which the information was obtained. VID represents the vehicle identifier. SPD, speed of the vehicle. XWAY on which freeway the car travels. LANE

Table 2
Input tuple schemas.

Input tuple	Schema
Car Position Data	(Time, VID, Spd, XWay, Lane, Dir, Seg)

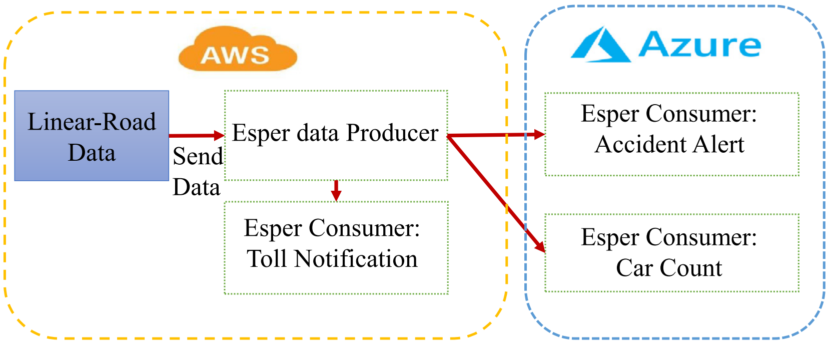
AWS
Linear-Road Data
Send Data
Esper data Producer
Esper Consumer: Toll Notification
Azure
Esper Consumer: Accident Alert
Esper Consumer: Car Count

Fig. 4. Simulation highway traffic pattern.

```
@name('toll-notification')
    Select vid, count(seg) as seg
    From CarLocStrEvent#time(30) group by vid;
```

Fig. 5. Toll notification query on Esper language.

Table 3
Output tuple schemas: continuous queries.

Query response	Schema
Toll-Notification	(VID, Seg)
Accident-Alert	(Xway, Lane, Seg)
Car-Count	(VID, XWay, Lane, Seg)

the road strip on which the car is. DIR, the direction, east or west. Finally, SEG represents the segment of the highway from which the position was issued.

The Esper language is based on the data-query pattern defined by SQL-92. For example, to define the toll collection notification (see Fig. 5), it used a grouping function that counted the number of segments (SEG) reported by the same vehicle in a 30 s time window. In the case of a same vehicle (IE VID) reporting a position of different segments within 30 s a toll collection event was triggered.

Following similar concepts, the car count query only counts the different VIDs, grouping these results by XWAY, LANE, and SEG. As well, the accident alert query counts the number of vehicles that have zero speed, grouping them by XWAY, LANE and SEG. When the number of vehicles with zero speed in the same tricycle: XWAY, LANE and SEG is greater than two, an accident alert event is generated as presented in Table 3.

5. Experimental evaluation

We conducted an experimental evaluation of the M2CPA monitoring system to evaluate its effectiveness and efficiency in monitoring cyber-physical applications running in multi-virtualizations deployed in multi-cloud environments. An application based on a highway data streaming system is deployed in a multi-cloud (Amazon and Azure) environment having both container and VM running it. We test our application by performing an extensive set of experiments using a 3 h data workload.

We considered both Amazon EC2 and Microsoft Azure clouds where we ran virtual machines using Ubuntu operating system (https://www.ubuntu.com/) 16.04 on which the Docker (https://www.docker.com/) platform, version 17, was installed to execute the application container. The VMs on Azure have the standard A1 configuration, with 1 VCPU and 2 gigabytes (GB) of memory for each machine, which consist of four VMs. The Amazons VMs were t2.micro instance, with 1 VCPU and 1GB of memory for each machine, which consist of two VMs.

The machine configurations on which experiments were conducted are as follows: first machine used Java (Version 8) virtual machine (used for S1). The second machine used Java (Version 8) virtual machine (used for S2). The third machine installed the Docker platform (version 1.18.0) and using Docker container that uses one image for Java to run (S3). The final machine used Java virtual machine and used this machine in Linear Road data producer to be consumed by S1, S2 and S3.

The application consisted of a cyber-physical system for monitoring highways. The sensed data (the position of the cars) is sent in a stream of events to be processed by three consumers: Toll Notification, Accident Alert and Car Count. The workload was composed of a file with 3 hours of heavy traffic. Three different scenarios covering different forms of virtualization

Table 4
Applications scenarios deployed at containers and VMS.

Environment	Scenario	Containers	VMs
Amazon Web Services (AWS) [A]	One-cloud Virtualization only (S1)		1- Linear-Road [A] 1- Toll-Notification [A]
Microsoft Azure Fabric [M] + Amazon Web Services (AWS) [A]	Multi-cloud Virtualization only (S2)		1- Linear-Road [A] 1- Car-Count [M]
Microsoft Azure Fabric blue[M] + Amazon Web Services (AWS) [A]	Multi-cloud Cross Container / VM (S3)	1- Accident-Alert [M]	1- Linear-Road [A]

in a multi-cloud environment (Amazon Web Services A and Microsoft Azure Fabric M) were proposed in order to obtain maximum reach for the various programming models of the cyber-physical system as shown in Table 4:

- Scenario 1 (S1) – A toll notification consumer and Linear road data producer running on the same cloud service. In our case, this was represented by the deployment of toll notification on the same cloud service as the Linear road data producer. Both were deployed on Amazon Web Services(A). The aim of the scenario was to understand the performance of applications running on the same cloud service.
- Scenario 2 (S2) – In this scenario we launch two virtual machines, one in each cloud. In Microsoft Azure Fabric (M), we run a car count consumer application. In Amazon Web Services (A), we run a Linear road data producer. The aim of the scenario was to understand the performance of the application running on multiple clouds.
- Scenario 3 (S3) – The last scenario serves as an evaluation of the type of virtualization (the data consumer is deployed in a Docker container). Within Microsoft Azure Fabric (M), we run accident alert in a container. In Amazon Web Services (A), we run a Linear road data producer. The aim of the scenario was to understand the performance of the application running on multiple clouds with multiple virtualization techniques.

We emphasize that the data load generated by the Linear road data producer was simultaneously sent to all three consumer applications within scenarios S1, S2 and S3.

5.1. CPU results

The CPU values for all scenarios is shown in Fig. 6. The monitoring agents send monitoring information to the manager every 5 s. As shown in Fig. 6 the average usage of CPU (%) for the toll notification service was 0.36%. For the accident alert service, in the Azure container, the average usage of CPU was 3.48%. However, the average usage of CPU for the car count service running in VM was 4.00%. The Linear road producer that was run in VM, and submitted in Amazon; had a bigger

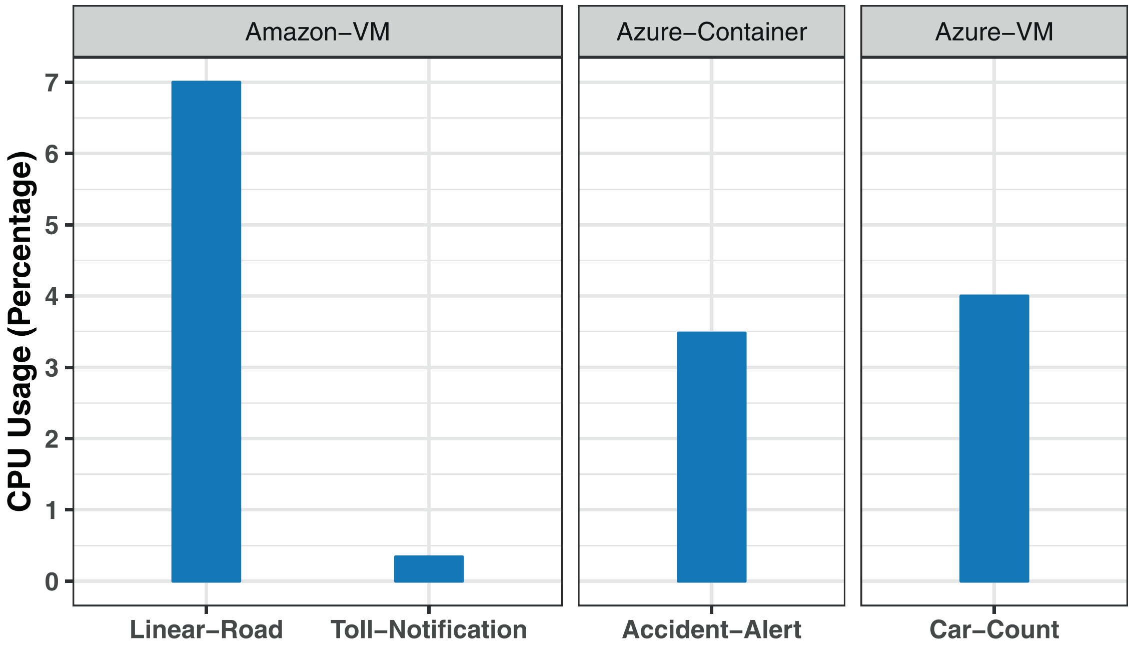

Fig. 6. CPU usage (Percentage) for services on VMs in Amazon, VM in Azure and container in Azure.

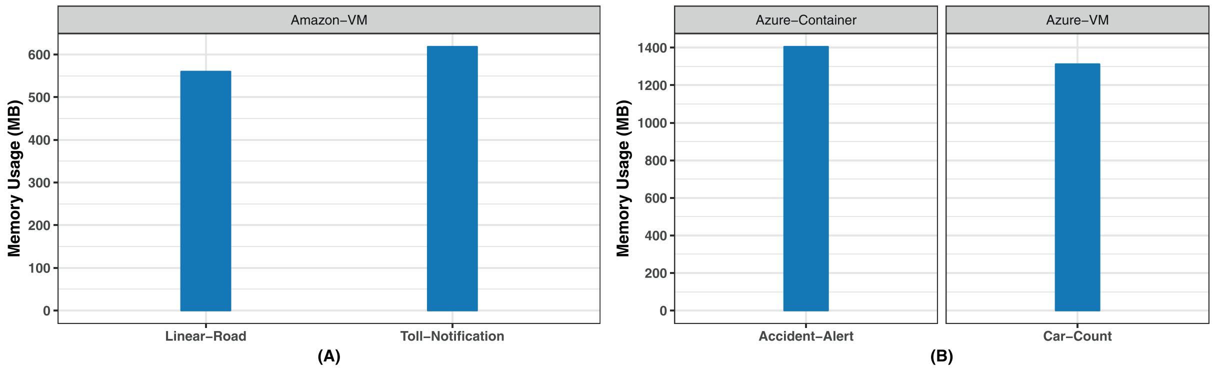

Fig. 7. Memory usage (MB) for services on VMs in Amazon (A), VM and container in Azure (B).

CPU usage of 7.00% because of continuous reading of 3 days worth of data from file and parsing this data using Esper event to be sent to all consumers.

5.2. Memory results

Fig. 7 shows memory usage results obtained from agents monitoring services running on both public clouds. The average memory usage for the toll notification service that is running in Amazon VM was 618MB from a memory total of 992MB as shown in Fig. 7(A). On the other hand, the memory usage on Azure is larger than on Amazon. The larger memory use on Azure cloud can be explained by the difference of virtual hardware configuration between the two clouds. When running a container in Azure, the average memory usage for the accident alert service was 1405MB as shown in Fig. 7(B). This is from a memory total within the container of 1920MB. Further, the average memory usage for the car count service running within a VM in Azure was 1312MB (as shown in Fig. 7(B)). This is from a memory total for the VM of 1936MB. The Linear road data producer was run in VM, and has an average of memory usage of 559MB as shown in Fig. 7(A). This is from a memory total for the VM of 992MB.

5.3. Network results

Fig. 8 shows network usage results obtained from agents monitoring the network traffic of the services. In the toll notification service, car count service, and the Linear road data service (workload of a file with 3 h of heavy traffic), the download and upload rates of a VM or container are presented. For the accident alert service the download and upload rate of the container are shown. The results show that the traffic caused by using a 3 h data workload, was detected and verified by the monitoring system. The network traffic of the toll notification service running in an Amazon cloud VM, was 495KB for

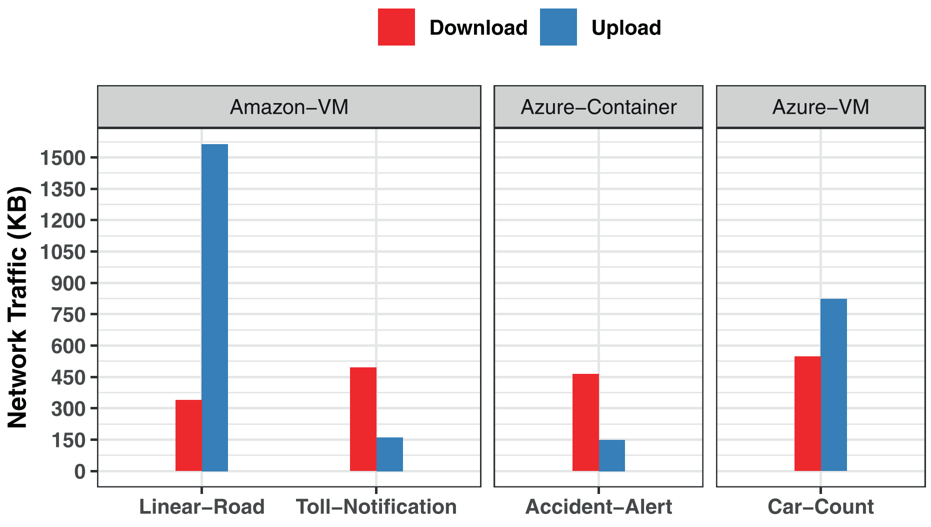

Fig. 8. Network traffic (KB) for services on VMs in Amazon, VM and container in Azure.

download and 161KB for upload. The network traffic of the accident alert service running in an Azure cloud container, was 464KB for download and 149KB for upload. The network traffic for the car count service running on an Azure cloud VM, was 548KB for download and 823KB for upload. The Linear road producer service had a network traffic of 399KB for download, and 1563KB for upload. This high upload is expected because it is sending the same data 3 times to all other services running on multiple clouds.

5.4. Results summary

In the previous sections we clearly see the effectiveness of our M2CPA framework in accurately monitoring the individual components of a cyber-physical application distributed across multiple clouds using multiple virtualisation means including VMs and containers. The M2CPA framework was able to calculate and report accurate performance metrics of CPU usage, memory usage, and network usage for 3 scenarios of a traffic monitoring application. Our work improves significantly on current monitoring tools in that it provides a unique combination of features that include a) monitoring the performance of cyber-physical application sub components running inside individual containers and individual VMs, b) gathers monitoring information from applications/sub-applications running inside heterogeneous cloud environments (e.g, Amazon, Azure, Open Stack, etc) and aggregates the results via an agent based system, c) stores monitoring data in a database shared by both containers and VMs, d) monitored data can be stored and accessed on any cloud provider.

6. Conclusion and future work

With the anticipated advent of new computing and networking technologies, we can expect to see billions of more devices being connected to the Internet as part of cyber-physical systems for critical applications such as smart healthcare, and smart cities. Developing reliable monitoring frameworks that can accurately assess the performance of such critical applications is extremely important. But with the number of components, and complexity of such applications expected to increase, monitoring their performance accurately and efficiently becomes more challenging. In this paper, we propose, develop and validate M2CPA – a novel framework for efficient monitoring of cyber-physical applications based on multi-virtualization (containers/VMs) and multi-cloud environments. The proposed solution provides users the ability to monitor the performance of cyber-physical applications that run inside containers and VMs and report their metrics performance in real-time. We developed a proof-of-concept implementation of the proposed solution using Docker containers and VMs deployed on Amazon and Azure clouds. The proposed system was evaluated using experimental analysis that considered diverse scenarios with evaluation outcomes validating the effectiveness of M2CPA in monitoring the performance of cyber-physical applications in a multi-virtualized and multi-cloud environment. Our future work will expand the framework to monitor physical devices and application container migration to develop efficient deployment and orchestration strategies for cyber-physical applications.

Declaration of Competing Interest

None.

Acknowledgements

This research is partially funded by two Natural Environment Research Council (UK) projects (FloodPrep:NE/P017134/1 and PACE: EPSRC EP/R033293/1).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.compeleceng.2019.06.007.

References

[1] Noor A, Jha DN, Mirta K, Jayaraman PP, Souza A, Ranjan R, et al. A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environment. In: IEEE International Conference on Cloud Computing (Cloud 2019), IEEE Computer Society; 2019.

[2] Yang LT, Wang X, Chen X, Wang L, Ranjan R, Chen X, et al. A multi-order distributed hosvd with its incremental computing for big services in cyber-physical-social systems. IEEE Trans Big Data 2018.

[3] Wang P, Yang LT, Li J, Chen J, Hu S. Data fusion in cyber-physical-social systems: state-of-the-art and perspectives. Inf Fusion 2019;51:42–57.

[4] Simmon E, Kim K-S, Subrahmanian E, Lee R, De Vaulx F, Murakami Y, et al. A vision of cyber-physical cloud computing for smart networked systems. US Department of Commerce, National Institute of Standards and Technology; 2013.

[5] Wang X, Wang W, Yang LT, Liao S, Yin D, Deen MJ. A distributed hosvd method with its incremental computation for big data in cyber-physical-social systems. IEEE Trans Comput Social Syst 2018;5(2):481–92.

[6] Kumar N, Zeadally S, Rodrigues JJ. Vehicular delay-tolerant networks for smart grid data management using mobile edge computing. IEEE Commun Mag 2016;54(10):60–6.

[7] Kumar N, Chilamkurti N, Misra SC. Bayesian coalition game for the internet of things: an ambient intelligence-based evaluation. IEEE Commun Mag 2015;53(1):48–55.

[8] Amin R, Islam SH, Biswas G, Khan MK, Kumar N. A robust and anonymous patient monitoring system using wireless medical sensor networks. Future Gen Comput Syst 2018;80:483–95.
[9] Zhang Q, Zhu C, Yang LT, Chen Z, Zhao L, Li P. An incremental cfs algorithm for clustering large data in industrial internet of things. IEEE Trans Ind Inf 2017;13(3):1193–201.
[10] Ardagna D, Casale G, Ciavotta M, Pérez JF, Wang W. Quality-of-service in cloud computing: modeling techniques and their applications. J Internet Serv Appl 2014;5(1):11.
[11] Alqahtani A, Li Y, Patel P, Solaiman E, Ranjan R. End-to-end service level agreement specification for iot applications. In: 2018 international conference on high performance computing & simulation (HPCS). IEEE; 2018. p. 926–35.
[12] Al-Ayyoub M, Jararweh Y, Daraghmeh M, Althebyan Q. Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure. Cluster Comput 2015;18(2):919–32.
[13] Li J, Hou M. Improving data availability for deduplication in cloud storage. Int J Grid High PerformComput (IJGHPC) 2018;10(2):70–89.
[14] Lee K, Gilleade K. Generic processing of real-time physiological data in the cloud. Int J Big Data Intell 2016;3(4):215–27.
[15] Yan H, Zhang Y, Wang H, Ding B, Mi H. S3r: storage-sensitive services redeployment in the cloud.. IJBDI 2017;4(4):250–62.
[16] Al-Shablan M, Tian Y, Al-Rodhaan M. Secure multi-owner-based cloud computing scheme for big data. Int J Big Data Intell 2016;3(3):182–9.
[17] Yang Y, Liu X, Zheng X, Rong C, Guo W. Efficient traceable authorization search system for secure cloud storage. IEEE Trans Cloud Comput 2018.
[18] Yang Y, Zheng X, Chang V, Tang C. Semantic keyword searchable proxy re-encryption for postquantum secure cloud storage. Concurrency Comput 2017;29(19):e4211.
[19] Pendlebury J, Emeakaroha VC, O'Shea D, Cafferkey N, Morrison JP, Lynn T. Somba-automated anomaly detection for cloud quality of service. In: Cloud computing technologies and applications (CloudTech), 2016 2nd international conference on. IEEE; 2016. p. 71–9.
[20] Li L, Tang T, Chou W. A rest service framework for fine-grained resource management in container-based cloud. In: Cloud computing (CLOUD), 2015 IEEE 8th international conference on. IEEE; 2015. p. 645–52.
[21] Alhamazani K, Ranjan R, Mitra K, Jayaraman PP, Huang Z, Wang L, et al. Clams: Cross-layer multi-cloud application monitoring-as-a-service framework. In: Services computing (SCC), 2014 IEEE international conference on. IEEE; 2014. p. 283–90.
[22] Jha DN, Garg S, Jayaraman PP, Buyya R, Li Z, Ranjan R. A holistic evaluation of docker containers for interfering microservices. In: 2018 IEEE international conference on services computing (SCC). IEEE; 2018. p. 33–40.
[23] Großmann M, Klug C. Monitoring container services at the network edge. In: Teletraffic congress (ITC 29), 2017 29th international, 1. IEEE; 2017. p. 130–3.
[24] Preeth E, Mulerickal FJP, Paul B, Sastri Y. Evaluation of docker containers based on hardware utilization. In: Control communication & computing India (ICCC), 2015 international conference on. IEEE; 2015. p. 697–700.
[25] Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, et al. Linear road: a stream data management benchmark. In: Proceedings of the thirtieth international conference on very large data bases-Volume 30. VLDB Endowment; 2004. p. 480–91.

Ayman Noor is a Ph.D. student in Computer Science of Newcastle University, UK. His current research interests include Cloud Computing and Monitoring. He earned a Master of Science in Computer and Information Science from Gannon University, PA, U.S.A in 2013 and a Bachelor in Computer Science from the College of Computer Science and Engineering from Taibah University, Madinah, SA in 2006.

Karan Mitra is an Assistant Professor at Luleå University of Technology, Sweden. He received his Dual-badge Ph.D. from Monash University, Australia and Luleå University of Technology in 2013. His research interests include quality of experience modeling and prediction, context-aware computing, cloud computing and mobile and pervasive computing systems. He is a member of the IEEE and ACM.

Ellis Solaiman is a Lecturer at the School of Computing, Newcastle University. He previously received his Ph.D. in Computing Science also from Newcastle University, where he subsequently worked as a Research Associate and Teaching Fellow. His research interests are mainly in the areas of Dependability and Trust in Distributed Systems such as the Cloud and the Internet of Things.

Arthur Souza is a Ph.D. student in the Post-Graduate Program in Information Sciences of the Federal University of Rio Grande do Norte, UFRN. He has experience in the areas of distributed systems, service-oriented architecture, cloud computing, IoT and distributed processing. He is currently participating in the Smart Metropolis Project and researches technologies focused on the context of intelligent cities.

Devki Nandan Jha is a PhD student in the School of Computing Science at Newcastle University, UK. His research interests include cloud computing, container, big data analytics, and Internet of Things. Jha has an MTech in Computer Science and Technology from Jawaharlal Nehru University, India.

Umit Demirbaga received the B.S. degree in Electronics and Computer Education from Marmara University, Turkey, in 2011, and the M.S. degree in Computer Science from Newcastle University, UK, in 2017. He is currently pursuing the Ph.D. degree in Computer Science with Newcastle University, UK. His current research interests include Cloud Computing, Parallel Computing and Big Data Processing.

Prem Prakash Jayarama is currently a Research Fellow at Swinburne University of Technology, Melbourne. His research areas of interest include, Internet of Things, cloud computing, mobile computing, sensor network middleware and semantic internet of things. Previously he was a Postdoctoral Research Fellow at CSIRO Digital Productivity Flagship, Australia from 2012 to 2015.

Nelio Cacho is an Associate Professor in computer science at the Federal University of Rio Grande do Norte, Brazil. He received his PhD in computer science from the University of Lancaster (2008). Cacho has worked in software engineering and distributed systems areas for the last 15 years.

Rajiv Ranjan is a Full professor in Computing Science at Newcastle University, United Kingdom. He was a Senior Research Associate (Lecturer level B) in the School of Computer Science and Engineering, University of New South Wales (UNSW). Prof. Ranjan has a PhD (2009) from the department of Computer Science and Software Engineering, the University of Melbourne.