Future Generation Computer Systems 99 (2019) 86–105

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

IoTSim-Stream: Modelling stream graph application in cloud simulation

Mutaz Barika [a,*], Saurabh Garg [a], Andrew Chan [b], Rodrigo N. Calheiros [c], Rajiv Ranjan [d,e]

[a] School of Technology, Environments and Design (TED), University of Tasmania, Australia
[b] School of Engineering, University of Tasmania, Australia
[c] School of Computing, Engineering and Mathematics, Western Sydney University, Australia
[d] School of Computing, Newcastle University, United Kingdom
[e] School of Computer Science, China University of Geosciences, Wuhan, PR China

HIGHLIGHTS

- Simulation model for stream graph application execution on the cloud.
- A novel XML-based structure for stream graph application specification.
- Model Multicloud environment.
- IoTSim-Stream for development of scheduling algorithms on Multicloud environment.

ARTICLE INFO

Article history:
Received 20 July 2018
Received in revised form 21 February 2019
Accepted 3 April 2019
Available online 9 April 2019

Keywords:
Internet of Things (ioT)
Stream processing
Stream graph applications
Multicloud environment
Simulator

ABSTRACT

In the era of big data, the high velocity of data imposes the demand for processing such data in real-time to gain real-time insights. Various real-time big data platforms/services (i.e. Apache Storm, Amazon Kinesis) allow to develop real-time big data applications to process continuous data to get incremental results. Composing those applications to form a workflow that is designed to accomplish certain goal is the becoming more important nowadays. However, given the current need of composing those applications into data pipelines forming stream workflow applications (aka stream graph applications) to support decision making, a simulation toolkit is required to simulate the behaviour of this graph application in Cloud computing environment. Therefore, in this paper, we propose an IoT Simulator for Stream processing on the big data (named IoTSim-Stream) that offers an environment to model complex stream graph applications in Multicloud environment, where the large-scale simulation-based studies can be conducted to evaluate and analyse these applications. The experimental results show that IoTSim-Stream is effective in modelling and simulating different structures of complex stream graph applications with excellent performance and scalability.

1. Introduction

In recent years, there is an emergence of "big data" term that has been introduced to deal with collecting, processing and analysing voluminous amount of data. It has three characteristics known as 3Vs of big data, which are volume (the size of data), variety (the different types of data collected) and velocity (the speed of data processing). To transact with big data, many big data platforms have been developed, which allow designing and building big data analysis applications to ingest, process as well as analyse tremendous amount of data. Composing those applications into data analysis pipeline forming big data workflow applications allows delivering of valuable analytical insights to make better decisions [1].

The execution and management of big data workflow application need a dynamic environment that provides the underlying infrastructure for big data processing, allowing parallel execution of this workflow application and to exploit large amount of distributed resources. As Cloud computing offers on-demand access to large-scale resources including compute, storage and network which can tackle extensive computation problems [2,3], it is seen as a visible solution for the execution of this workflow application. Even more, the Multicloud environment that consolidates multiple Clouds is more visible solution for orchestrating the execution of multiple applications included in such workflow application over various Clouds. Other than utilizing these

* Corresponding author.
E-mail addresses: mutaz.barika@utas.edu.au (M. Barika), saurabh.garg@utas.edu.au (S. Garg), andrew.chan@utas.edu.au (A. Chan), R.Calheiros@westernsydney.edu.au (R.N. Calheiros), Raj.Ranjan@newcastle.ac.uk (R. Ranjan).

https://doi.org/10.1016/j.future.2019.04.004
0167-739X/

resources, the requirements of big data workflow applications such as near real-time data analysis need to be ensured. The requirement of orchestration systems that can help in execution and management of big data workflow applications on a Cloud and Edge infrastructure is pointed out by [1] as the most important and cutting edge research issue. Accordingly, the need of understanding the behaviour of these applications when they are executing in Cloud environment and for development of new scheduling and resource provisioning techniques is important to ensure that requirements of these applications can be successfully met while utilizing Cloud infrastructure efficiently and effectively.

Studying how big data workflow applications will perform in the Cloud and evaluating the efficiency of new scheduling and resource allocation algorithms for such applications currently is not an easy task. These problems are often hard to be investigated on real-world Cloud infrastructures due to the following reasons: (1) unstable and dynamic nature of Cloud resources, (2) scalability and complex requirements of stream workflow application, and (3) real experiments on large, heterogeneous and distributed Cloud platforms are subject to the impact of external events, notably not cost-effective, considerably time-consuming and different conditions cannot be reproducible to easily reproduce results. The visible approach for evaluating application benchmarking study in repeatable, controllable, dependable and scalable environments is via simulation toolkits, where experimental results can be reproduced easily [4]. Therefore, a simulator supporting stream graph application is a very useful software toolkit, allowing both researchers and commercial organizations to model their stream graph applications and evaluate the performance of their algorithms in heterogeneous Cloud infrastructures at an effective time and with no cost.

To address the above research problems, we design and implement an IoT Simulator for Stream graph applications (IoTSim-Stream) that extended a popular and widely used Cloud computing simulator (CloudSim), where we model stream workflow application in Multicloud environment. It provides the ability to model and simulate the execution of stream graph application over resources provisioned from various Cloud infrastructures. In summary, the following are our contributions:

- Modelling stream graph application.
- Extending the XML structure of commonly existing non-streaming workflow strictures (e.g. Montage, CyberShake) to simulate stream graph applications.
- Modelling Multicloud environment as an execution environment for stream graph application.
- Proposing a new simulator named IoTSim-Stream that leverages the features of CloudSim and integrating real-time processing model with workflow scheduling and execution to execute the modelled stream graph application in Multicloud environment.

This paper is structured as follows: Section 2 describes what stream graph application is, while Section 3 outlines design issues of this type of workflow application. Section 4 reviews the related simulation tools. Section 5 presents the architecture of the proposed simulator (IoTSim-Stream), while in Section 6, we explain in detail the implementation of IoTSim-Stream including the extended XML structure, proposed provisioning and scheduling policy, proposed stream scheduling policy and proposed VM-level scheduler. Section 7 presents our experiments to validate and evaluate the performance and scalability of IoTSim-Stream in simulating stream graph applications in Multicloud environment, and discusses the obtained results. Section 9 concludes the paper and highlights future improvements.

2. Stream graph application

Stream Graph application is a network of streaming data analysis components, where each individual component can be considered as a service and is executed independently over compute resources that provisioned from the Cloud, even though data dependencies among services should be maintained. Fig. 1 presents an example of stream graph application with its data processing requirements. The execution of this type of workflow application is continuous (i.e. not one-time execution). It starts when the data streams generated by external sources such as sensors being continuously injected into data pipeline (particularly as input data streams to services). The data processing on these input data streams is continuously carried-out by those services to produce continuous output data streams (i.e. online insights) that are results of data processing computations. These output data streams generated by internal sources (i.e. parent services) are continuously injected into data pipeline, specifically as input data streams to child services, which process them continuously and then inject the results of computations into data pipeline. Therefore, we simply can say that this graph application has three main characteristics: continuous input data streams from external sources towards connected services and from internal sources (as results of computations that routed from these internal sources (i.e. parent services) to child services), continuous data processing of input data streams and continuous output data streams that are results of data processing computations at graph services.

As noted in Fig. 1, each service has data processing requirement (the number of instructions required to process one MB of data stream) and data processing rate (the amount of streaming data the service can process per second such as 30MB/s). The owner of stream graph application can define user specific performance constraints in term of data processing rates on services, where these constraints are always maintained during the execution of this application. In case of no user performance constraint is specified on the service or the value defined is less than the speed of incoming streams, the total size of incoming streams for this service will be considered as a performance constraint. During the continuous execution of stream graph application, each service receives streaming input data from external sources and/or internal sources (i.e. parent services), processes them continuously as they arrive and generates streaming output data as results of computations which routed towards one or more child services based on the specified data modes (replica or partition). With replica mode, the output stream of parent service is replicated on child service(s) while with partition mode, the output stream of parent service is partitioned into portions based on the pre-defined partition percentages and then each portion is routed to corresponding child service. The end service(s) produces streaming output results for the execution of this graph application.

3. Design issues of stream graph application

Unlike batch-oriented data processing model that intends to process static data (i.e. the amount of input data is finite and it is stored before being processed and analysed) [5,6], stream-oriented data processing model is intended for processing continuous data to gain immediate analytical insights. With this model, data arrives in streams, which are assumed to be infinite and are being processed and analysed (in a parallel and distributed manner) as they arrive and as soon as possible to produce incremental results at the earliest they are prepared [5,6]. Based on this model, stream applications have been developed to process continuous data to produce continuous analytical results. However, given

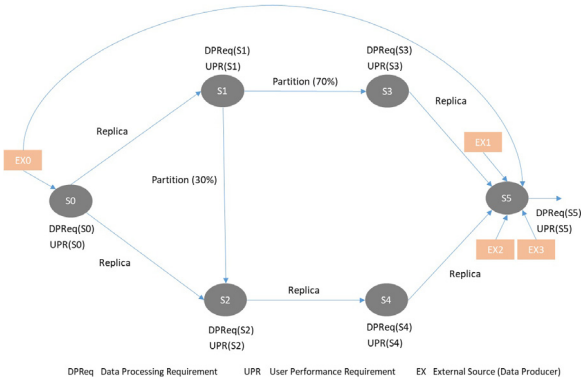


Fig. 1. Sample stream graph application.

the demand of composing those applications into data pipelines forming stream graph application, this graph application has specific design issues. In the subsequent paragraphs, we will review these issues.

Modelling of graph nodes. — Streaming data applications included in stream graph application can be considered as services since they can be separately running over any virtual resources, even though the data dependencies among them should be maintained. These independent processing nodes are allocated to appropriate VMs according to their performance requirements for processing continuous streams of data and producing analytical insights at the earliest they are prepared. Therefore, the simulator should models the nodes of graph as services adhering the data dependencies among them.

Modelling of data flows. — The flow of data in this type of workflow application is streams, which are infinite continuous events. These streams are continuously injected as inputs into nodes (services) and continuously produced as results of computations, i.e. outputs of nodes (services). The simulator should thus represent this type of data as a sequence of events and allow transmitting them among VMs hosted by various datacentres.

Synchronization of data flows. — In stream graph application, there exists data flow dependencies across analytical nodes, resulting in the need of data flow synchronization. Therefore, the execution of nodes (services) requires dynamic synchronization of the states (e.g. output stream of parent service forms the basis of input data stream to one or more child services) of parent and child services. Hence, the simulator should preserve the synchronization of data flows as it directly impacts the correctness of stream graph application execution.

Modelling of Multicloud environment and its network performance. — As the execution of stream graph application will be carried-out over multiple Cloud infrastructures, the simulator should model Multicloud environment as an execution environment for this application. Not only this, but also the execution of this application on resources provisioned from multiple Clouds means by the way that the streams of data are being transferred between VMs of datacentre (inbound traffic) or between different VMs hosted by various Cloud datacentres (outbound traffic). Therefore, the simulator also requires to model the inbound and outbound network performance (i.e. bandwidth and latency) between Cloud datacentres being used during simulation runtime, as the amount of streams being transferred is subject to the availability of bandwidth and the amount of delay.

4. Related simulation frameworks

With the emerging of Cloud computing, various simulation-based toolkits have been developed in order to model the behaviour of different Cloud services and applications on Cloud infrastructures. These simulators help researchers in evaluating the performance of these systems and applications in controllable environment.

To the best of our knowledge, there is no simulator that model the execution of stream graph application in various resources provisioned from multiple Cloud infrastructures. The most related simulators proposed by previous research works are described in the below paragraphs.

CloudSim [4]. — It is a popular and widely used event-based simulator that models and simulates Cloud computing infrastructures, applications and services. As an extensible and customizable tool, it allows to model custom Cloud application services, Cloud environments and application scheduling and provisioning techniques. In this simulator, users create Cloud tasks (named Cloudlets) to define their workloads and then submit them to Virtual Machines (VMs) provisioned from Cloud datacentre to be processed in the Cloud. The application model of CloudSim is simpler and is more appropriate to simulate batch tasks, so that it is not capable to support stream tasks (i.e. continuous computation).

NetworkCloudSim [7]. — It is a simulation toolkit that models Cloud datacentre network and generalizes applications (e.g. High Performance Computing and e-commerce). It allows computational tasks involved in these applications to communicate with each other. NetworkCloudSim supports advanced application models and network model of datacentre, allowing researchers to accurately evaluate the new scheduling and provisioning techniques in order to enhance the performance of Cloud infrastructure. Despite the advanced application models (i.e. multi-tier web application, workflow and MPI) supported by this simulator, the lack of application model that describing big data workflow applications is a major drawback in this simulator. Thus, it does not have the capability to simulate stream tasks and even execute stream workflow applications in Cloud environments.

MapReduce simulators (MRPerf [8], Mumak [9,10], SimMR [11], MRSim [12] and MR-CloudSim [13]). — MRPerf [8] is phase-level simulator for MapReduce processing model. It serves as a design tool for analysing MapReduce based applications performance on specific configurations of Hadoop system, and as a planning tool for evaluating the proposed designs and topologies of cluster. Mumak [9,10] is an Apache discrete event simulator for MapReduce verification and debugging. It takes as input the job trace data from real experiment along with the definition of cluster and then feeds them into simulator to simulate the execution of jobs in the defined virtual cluster with various scheduling policies. SimMR [11] is MapReduce based simulator developed in HP lab. It takes as input the execution traces derived from production workloads and then replies them to facilitate performance analysis and evaluating of new scheduling algorithms in MapReduce platforms. MRSim [12] is a discrete event simulation tool that extends SimJava, a Java discrete event engine to simulate various types of MapReduce-based applications and uses GridSim for network simulation. It offers functionalities for measuring the scalability of MapReduce applications and studying the effects of various Hadoop setup configurations on the behaviour of these applications. MR-CloudSim [13] is a simulator tool for modelling MapReduce based applications in Cloud computing environment. It is extended the feature of CloudSim to implement bare bone structure of MapReduce on CloudSim, supporting data processing

operations with this model. Thus, MR-CloudSim provides the ability for examining MapReduce Model in a Cloud-based datacentre. However, these simulators are only intended to support data processing operations with MapReduce model, thus they lack of support for modelling the streaming big data applications and even streaming big data workflow applications.

IoTSim [14]. — It is a software toolkit that built on top of CloudSim to simulate Internet of Things (IoT) applications in the Cloud infrastructure. It integrates IoT application model to allow processing of IoT data by the use of big data processing platform in Cloud infrastructure, providing both researchers and commercial entities with the ability to study the behaviour of those applications in controllable environment. This simulator is intended to support IoT application with MapReduce model, where it lacks the support for stream computing model. Therefore, it neither simulate stream big data application nor stream workflow application.

CEPSim [15]. — It is a simulator for event processing and stream processing systems in the Cloud computing environment. It uses query model to represent user-defined query (application), where the modelled query (with all its vertices) is allocated to a VM to be simulated at once. With such simulator and by default, users have to determine manually the placements of their queries when submitting them to CEPSim. Therefore, the main drawbacks of this simulator are (1) the user-defined query is executed entirely in a single VM, (2) provisioning resources according to input event streams of query is missing and (3) mapping of vertices to VMs is manual.

WorkflowSim [16]. — It is a simulation toolkit that extends CloudSim to support scientific workflow scheduling and execution in the Cloud with consideration of system overheads and failures. It incorporates model of workflow managements systems (similar to Pegasus workflow management system) in the Cloud simulation environment, enabling researchers to study and evaluate the performance of workflow optimization algorithms and methods more accurately. This simulator is intended to support scientific workflow applications, where it lacks the support for big data workflow applications (batch, stream or hybrid). Therefore, it neither simulate streaming big data applications nor streaming big data workflow applications.

Additionally, the common/shared drawback with all comparable simulators mentioned above is that they do not leverage the advantages of Multicloud environment to execute the modelled application on resources provisioned from various Cloud infrastructures, where the proposed simulator supports that. This will open the door for further research studies including proposing resource and scheduling policies, improving performance and minimizing execution cost. The summary of the above mentioned simulators along with their strengths and weaknesses are provided in Table 1.

5. The proposed architecture of IoTSim-stream

The CloudSim is a simulation framework that models and simulates Cloud infrastructures and services [4]. It has rich features that make it the best choice to be the core simulation engine for our proposed simulator to simulate the behaviour of stream graph applications and their execution in Multicloud environment. Fig. 2 shows the layered architecture of CloudSim with the essential elements of IoTSim-Stream (shown by orange-outlined boxes). In the subsequent paragraphs, we will describe these layers.

CloudSim core simulation engine layer. This layer takes care of the interaction among the entities and components of CloudSim via message passing operations [7]. It offers numerous key functions e.g. events queuing and handling, Cloud entities creation (such as datacenter, broker), entities communication, and simulation clock management [14]. Entity within the ambit of the CloudSim is a component instance, which could be either a class or group of classes that depicts one CloudSim model (datacenter, broker) [4]. It individually and independently exists, and has the capability for sending and receiving events to and from other CloudSim entities as well as process the received ones [14]. Event is a simulation event or message that passes among the CloudSim entities and holds relevant information e.g. the type of event, time at which this event occurs as well as the data passed in this event to destination entity [14].

CloudSim simulation layer. This layer is designed to model the core elements of Cloud computing. It contains several sub-layers to achieve that. The Network sub-layer models the topology of network among various datacentres, while Cloud Resource sub-layer models datacentre and Cloud coordinator, thereby these components of those sub-layers allow to design IaaS environments [14]. The Cloud and VM Services sub-layers offer the functionality required for designing VM management and scheduling algorithms for Cloud applications [14]. The sub-layer above, User Interface Structures, allows users to implement their structures for VM, Cloud application and application cloudlet.

Service layer. This layer concentrates on orchestrating the execution of streaming data applications included in stream graph application.

User code layer. This layer consists of two sub-layers, Scheduling Policy and Simulation Specification, providing the ability for users to specify their simulation configurations and scenarios in order to validate their scheduling and provisioning algorithms [7].

The descriptions of IoTSim-Stream elements are as follows:

- Graph Application — It is a Directed Acyclic Graph (DAG) that represents a graph application.
- Graph Application Configuration — It defines simulation runtime, application and user requirements.
- Graph Application Engine (GraphAppEngine) — It parses DAG input file and handles the whole execution process of graph application. This process includes provisioning VMs from different providers, scheduling services of graph application on the provisioned VMs and the submission of graph application cloudlets to those VMs.
- Graph Application Cloudlet (GraphAppClouldlet) — It represents a graph application with multiple stream application nodes (i.e. services).
- ServiceCloudlet — It represents a generalized stream application node.
- Graph Application Cloudlet Execution — It executes the submitted cloudlet (i.e. ServiceCloudlet) on VM.
- Big Datacenter (BigDatacenter) — It represents a Cloud resource whose has a list of virtualized hosts, offers various flavours of VM, where the provisioned VMs are allocated on these hosts.
- Stream VM (SVM) — It represents a Cloud resource where the mapped ServiceCloudlet will be executed on it.

6. Implementation

As we mentioned before, the proposed simulator (IoTSim-Stream) extends CloudSim with new functionality to support modelling the execution of stream graph application in multiple Cloud infrastructures. In line with the aforesaid design issues

Table 1
Summary of related simulators.

Simulator	Core engine	Pl	Strengths	Weaknesses
CloudSim [4]	GridSim	Java	• Model IaaS Cloud and batch tasks (long-running tasks) • Pluggable VM and application scheduling policy • Support of federated Cloud environment	• Lack of modelling application models that have communicating tasks [7] • Limited network support [7]
NetworkSim [7]	CloudSim	Java	• Model parallel applications such as (multi-tier web application, workflow and MPI) • Full network support (network-packet level) • Customize type of switches (root, aggregate and edge switch)	• Lack of big data applications support • Lack of support for stream tasks and even stream workflow applications • No Multicloud support
MRPerf [8]	CloudSim	Mix of C++, Tcl and Python	• Model big data batch processing (MapReduce) • Capture behaviour of Hadoop cluster • Simulate the full network by relying on ns-2	• Limited application behaviour (job has simple map and reduce tasks [11]) • Lack of stream / real-time processing support
Mumak [9] [10]	Discrete event simulator engine	Java	• Verify/debug Hadoop MapReduce framework • Perform no actual I/O or computations • Simulate behaviour of production cluster	• No modelling of shuffle/sort phase [11] • No simulating of Cloud resources • No job dependency • No modelling of failure correlations (only task-level failures) • Lack of stream processing support
SimMR [11]	Discrete event simulator engine	Not available	• Simulate MapReduce applications • Replayable MapReduce workload • Pluggable scheduling policy	• Lack modelling of Cloud • Lack of stream processing support
MRSim [12]	SimJava and GridSim package	Java	• Simulate Hadoop environment • Model shared Multi-core CPUs, HDD, and network topology and traffic • Consider cluster configurations	• Limited network support • Inherited limitations of SimJava (such as no support to create new simulation entity at runtime) • Lack of stream processing support
MR-CloudSim [13]	CloudSim	Java	• Model MapReduce-based applications	• Single-state map and reduce computation [14] • Limited network support (no network link modelling) [14] • No support to allow multiple MapReduce-based applications [14] • Lack of stream processing support
IoTSim [14]	CloudSim	Java	• Model IoT application with MapReduce model • Allow to simulate multiple IoT applications • Model network and storage delays incurred during the execution of IoT-based applications	• Lack of stream processing support • Lack of big data workflows support
CEPSim [15]	CloudSim	Scala and Java	• Model event processing queries • Customize operator placement, scheduling and load shedding strategies	• Limited network support • Query is executed entirely in a single VM • Provisioning resources according to input event streams of query is missing • Manual mapping of vertices to VMs
WorkflowSim [16]	CloudSim	Java	• Model scientific workflows • Consider diverse system overheads and failures	• No simulation of stream workflow applications

and requirements, the implementation of this simulator consists of two parts, which are modification and addition. The modification part is to modify the original code of CloudSim components such as datacenter and VM. While addition part is to add more components to meet the new requirements such as GraphAppEngine.

Fig. 3 shows the class diagram of IoTSim-Stream. The components with orange-outlined boxes as shown in this figure can be classified either into an entity or a class as follows:

- Main entities
  - GraphAppEngine: It extends SimEntity to handle the execution of stream graph application. That is including workflow provisioning and scheduling, Data Producers (DPs) starting-up and shutting-down, and simulation shutting-down based on pre-defined simulation time.
  - BigDatacenter: It extends native Datacenter, which is an SimEntity, to support simulation of stream graph application that includes handling of VMs and transferring streams in between VMs and out of this datacentre to other datacentres.
  - External Source: It extends SimEntity to represent any king of DP connected to the data source such as sensor, device or application and generates a continuous data stream.
- Classes for modelling Multicloud environment
  - VMOffers: It is an abstract class that encapsulates VM instance options offered by different Cloud service providers such as Microsoft Azure, Amazon EC2 and Google Compute Engine. Each implementation of this abstract class represents the VM options offered by a particular Cloud provider.
  - VMOffersBigDatacenter: It extends VMOffers abstract class to encapsulates different VM options offered by a particular Cloud provider (i.e. a BigDatacenter).

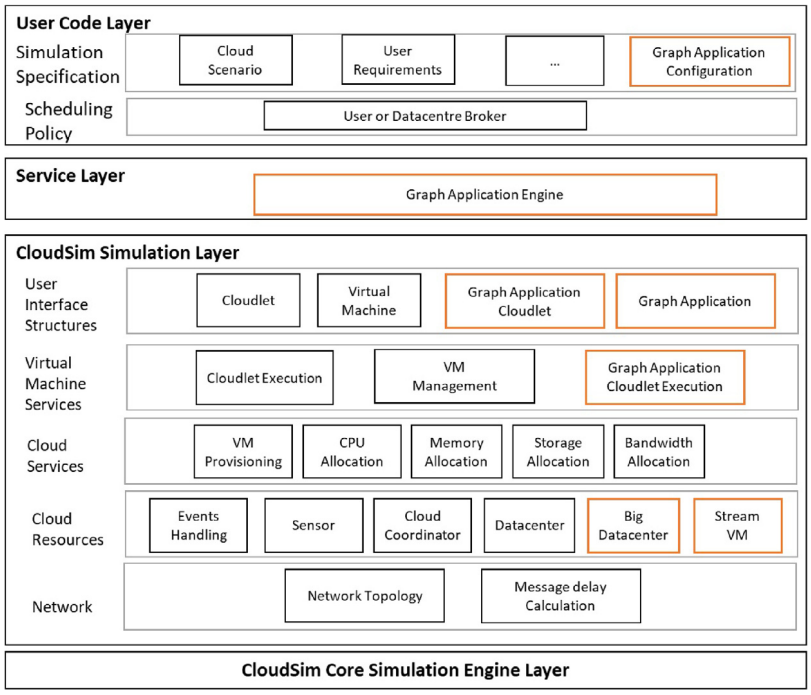


Fig. 2. The proposed architecture of IoTSim-Stream (CloudSim with IoTSim-Stream elements). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

- SVM: It is an extended class of the core VM object to model a VM with input and output stream queues, to be a Stream VM.
- ProvisionedSVM: It is a class designed to encapsulate a provisioned SVM with its information including the start and end time, and the cost.

- Classes for modelling basic BigDatacenter network
  - Channel: It is a class designed to represent a channel, which can be either ingress channel for transmitting streams between SVMs located at the same datacentre or egress channel for transmitting streams among SVMs located at different datacentres. It controls the amount of data transmitted in a shared data medium. Each channel, whether ingress or egress, is a shared channel among different simultaneous stream transmissions (time-shared mode).
  - StreamTransmission: It is a class represents transmission of a stream from source SVM to destination SVM located at the same datacentre or at different datacentres.

- Classes for modelling stream graph application
  - GraphAppCloudlet: It is a class designed to represent a stream graph application with multiple graph nodes i.e. services as described in the XML file of this graph application.
  - Service: It is a class designed to model atomic node in stream graph application as a service that processes incoming data stream(s) and produce output stream. It contains service information including service identification, data processing requirement, user performance requirement, its ServiceCloudlets, dependencies streams, parent service(s), child service(s) and output stream.
  - ServiceCloudlet: It is an extended class of the core Cloudlet object to implement an atomic graph node, which will be submitted to the Cloud datacentre (i.e. BigDatacenter) by GraphAppEngine and executed in SVM. The atomic graph node or service can be modelled using one or more ServiceCloudlets. That is allowing parallel execution of service computations, and enhancing scalability and overall execution performance while meeting user performance requirements easily. Of course, each ServiceCloudlet contains the information of service to which it belongs.
  - Stream: It is a class designed to model data unit that being processed in this simulator. This class is used to represent both stream and stream portion when the original stream splits into several portions.

- Classes for scheduling ServiceCloudlets
  - Policy: It is an abstract class that implements the abstract policy for provisioning resources and scheduling of stream graph application (represented in DAG) in an IaaS datacentre. This class performs common tasks such as parsing the XML file describing the DAG, printing the scheduling plan, and returning provisioning and scheduling decisions to the GraphAppEngine.
  - SimpleSchedulingPolicy: It is an extended class from policy abstract class that represents the implementation of simple provisioning and scheduling policy for stream graph applications. It is first responsible for selecting the most suitable SVMs for each service whose achieved user performance requirement, and then scheduling the ServiceCloudlets of this service on them for execution. The detailed of this scheduling policy that offered in our simulator will be discussed in the next section.
  - ServiceCloudletScheduler: It is an extended class of the core CloudletScheduler object to implement a space shared scheduling policy performed by SVM to run ServiceCloudlet. The detailed of this scheduler will be discussed in the next section.

- Class for scheduling streams on SVMs
  - StreamSchedulingOnSVMs: It is a class designed to schedule the divided portions of each stream either input or output stream on SVMs of destination service according to their computing power.
- Classes for customizing simulation parameters
  - Properties: It is an enumeration class represented the customizable parameters from simulation that are defined in simulation properties file (named simulation.properties).
  - Configuration: It is a class implements properties manager, which loads simulation properties file (i.e. simulation.properties) contained parameters of simulation that are customized by users.

6.1. Extending XML structure of synthetic workflows

Common workflow structures from different application domains [17], such as Montage in Astronomy, Inspiral in Astrophysics, Epigenomics in Bioinformatics and CyberShake in Earthquake science, operate on static data inputs and produce final outputs. Therefore, the XML structure generated by a set of synthetic workflow generators is describing these static workflows and its parameters. However, the use of these workflow structures to simulate stream graph applications is practically feasible, as each job is considered a service and the data flow becomes streams of data. The inputs of a job incoming from static files (not from parent jobs) become the continuous inputs of a service from DPs (i.e. external sources). The service continuously processes incoming data streams and continuously produces output stream. The output of a parent job, which is sent to one or more child jobs, becomes the continuous output of a parent service that is sent to one or more child services.

Accordingly, there is a need for extending the original structure of those workflows to describe the additional parameters and attributes of stream graph applications such as data processing requirements, input and output data rates. By making this extension, those workflow structures become stream graph structures. Table 2 lists the parameters and attributes being used in the extended XML structure.

To aid understanding how to describe stream graph application in the extended XML structure using the above mentioned parameters and attributes, we use the presented sample stream graph application in Fig. 1 and depict its XML structure in Listing 1.

Listing 1: Extended XML structure of sample stream graph application

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2018-02-27:11:00 -->
<!-- generated by: Mutaz -->
<adag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
      count="6" name="SampleStreamGraphhApplication" serviceCount="6"
      childCount="5">
<!-- part 1: list of all referenced outputs of services (may be empty) -->
<!-- part 2: definition of all services (at least one) -->
 <externalsources>
  <exsource id="PID00000" name="Producer0" type="stream" datarate="10"/>
  <exsource id="PID00001" name="Producer1" type="stream" datarate="10"/>
  <exsource id="PID00002" name="Producer2" type="stream" datarate="5"/>
  <exsource id="PID00003" name="Producer3" type="stream" datarate="5"/>
  <exsource id="PID00004" name="Producer4" type="stream" datarate="5"/>
 </externalsources>
 <service id="ID00000" dataprocessingreq="400" userreq="10" namespace="
      Sample" name="BigService0" version="1.0">
  <uses link="input" type="stream" producerref="PID00000"/>
  <uses link="output" type="stream" size="5"/>
 </service>
 <service id="ID00001" dataprocessingreq="1000" userreq="5" namespace="
      Sample" name="BigService1" version="1.0">
  <uses link="input" type="stream" processingtype="replica" serviceref="
      ID00000"/>
  <uses link="output" type="stream" size="10"/>
 </service>
 <service id="ID00002" dataprocessingreq="500" userreq="8" namespace="Sample
      " name="BigService2" version="1.0">
  <uses link="input" type="stream" processingtype="replica" serviceref="
      ID00000"/>
  <uses link="input" type="stream" processingtype="partition"
      partitionprecentage="30" serviceref="ID00001"/>
  <uses link="output" type="stream" size="8"/>
 </service>
 <service id="ID00003" dataprocessingreq="2000" userreq="7" namespace="
      Sample" name="BigService3" version="1.0">
  <uses link="input" type="stream" processingtype="partition"
      partitionprecentage="70" serviceref="ID00001"/>
  <uses link="output" type="stream" size="1"/>
 </service>
 <service id="ID00004" dataprocessingreq="3000" userreq="8" namespace="
      Sample" name="BigService4" version="1.0">
  <uses link="input" type="stream" processingtype="replica" serviceref="
      ID00002"/>
  <uses link="output" type="stream" size="2"/>
 </service>
 <service id="ID00005" dataprocessingreq="1500" userreq="38" namespace="
      Sample" name="BigService5" version="1.0">
  <uses link="input" type="stream" producerref="PID00000"/>
  <uses link="input" type="stream" producerref="PID00001"/>
  <uses link="input" type="stream" producerref="PID00002"/>
  <uses link="input" type="stream" producerref="PID00003"/>
  <uses link="input" type="stream" producerref="PID00004"/>
  <uses link="input" type="stream" processingtype="replica" serviceref="
      ID00003"/>
  <uses link="input" type="stream" processingtype="replica" serviceref="
      ID00004"/>
  <uses link="output" type="stream" size="4"/>
 </service>
 <!-- part 3: list of control flow dependencies (may be empty) -->
 <child ref="ID00001">
  <parent ref="ID00000"/>
 </child>
 <child ref="ID00002">
  <parent ref="ID00000"/>
  <parent ref="ID00001"/>
 </child>
 <child ref="ID00003">
  <parent ref="ID00001"/>
 </child>
 <child ref="ID00004">
  <parent ref="ID00002"/>
 </child>
 <child ref="ID00005">
  <parent ref="ID00003"/>
  <parent ref="ID00004"/>
 </child>
</adag>
```

6.2. Stream scheduling

Since achieving user-defined performance requirement for a service may need more than one SVMs, this service will need more than one ServiceCloudlets, where each one is mapped to one SVM, leading to this service being mapped to more than one SVMs. Therefore, the incoming data streams from external sources and parent services towards this service should be divided into portions and distributed across its SVMs according to their computing power. Similarly, the output data stream producing by parent service towards child service(s) should be divided into portions and sent to the SVM(s) provisioned for such child service.

Consequently, we implement stream scheduling policy defined in the StreamSchedulingOnSVMs Java class. It divides each data stream into portions and schedules them in round-robin fashion according to computing power of SVMs of destination service. For instance, if one of child services in stream graph application has two SVMs, where the computing power of first VM is twice computing power of the second one, the divided portions of one output stream of parent service are distributed into 2:1 way — two portions for first VM and one portion for second VM.

6.3. Scheduler and execution of ServiceCloudlet

Before providing the details of the implemented scheduler in IoTSim-Stream, we need to discuss how IoTSim-Stream is initializing and what is the provisioning and scheduling policy being used to schedule stream graph application on Multi-cloud environment. Algorithm 1 shows the pseudo-code of simple provisioning and scheduling algorithm that we implemented in IoTSim-Stream. This algorithm provisions the most suitable VMs for services included in stream graph application which meet the user performance requirements for those services, where all VMs

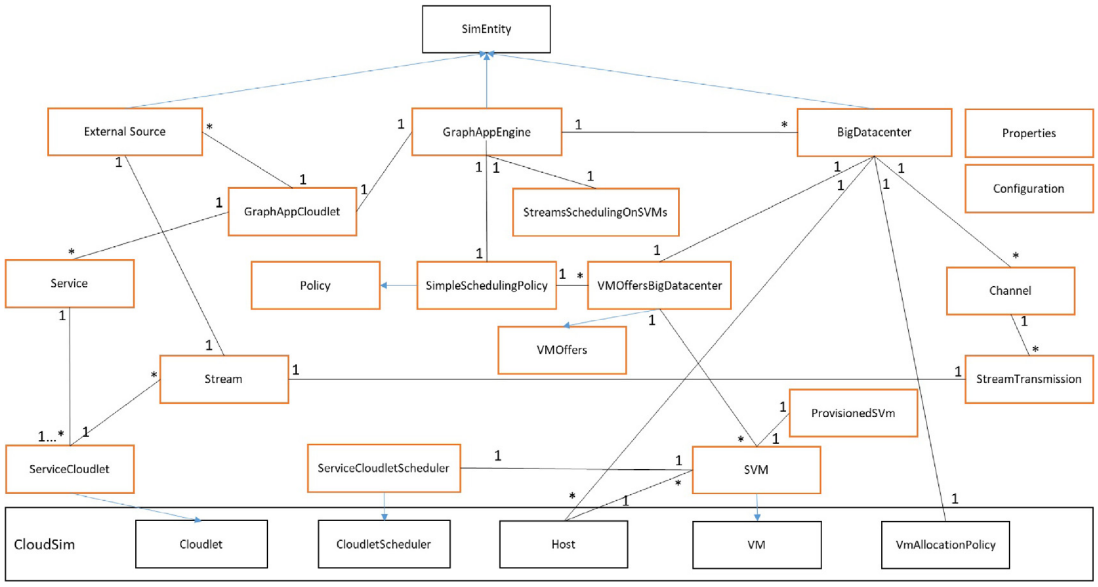


Fig. 3. Class diagram of IoTSim-Stream. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Additional parameters of stream graph application.

Parameter	XML attribute name	Data type	Value
Data Processing Requirement	dataprocessingreq	Integer	ex. 1000 (in MI/MB)
User Performance Requirement	userreq	Number	ex. 10 (in MB/s)
Reference Input from Parent Service	serviceref	String	Referenced id of parent service as defined in XML file (ex. ID00001)
Processing Type of Input from Parent Service	processingtype	String	replica or partition
Partition Processing Type for Input Stream	partitionprecentage	Integer	1–99
External Source Identifier	id	String	ex. PID00000
External Source Name	name	String	ex. Producer0
External Source Data Rate	datarate	Number	ex. 12.5 (in MB/s)
Reference Input from External Source	producerref	String	The referenced id of external source as defined in XML file (ex. PID00000)
Service Output Data Rate	size	Number	ex. 20 (MB/s)

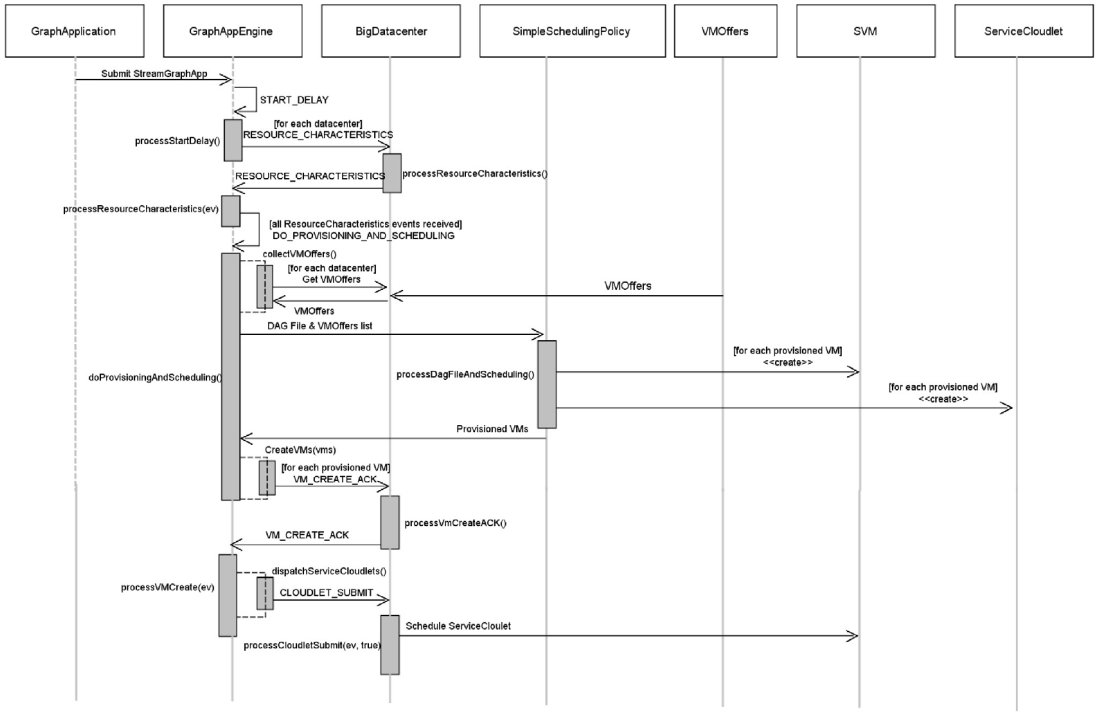


Fig. 4. Sequence Diagram: The flow of communication for initializing IoTSim-Stream and scheduling ServiceCloudlet on SVMs.

for a service are provisioned from one Cloud-based datacentre. For each service, it finds VMs with higher computing powers upto required MIPS value (that is calculated based on user performance requirement and service processing requirement) and provisions them to achieve as much as possible from this value. Then, it backs to VMs with lower computing powers to achieve the remaining value. Nevertheless, in case of the selected VMs list for any service is empty, IoTSim-Stream shows a message to the user indicating that, and then is terminated. This happens because there is no VM offer available in the selected datacentre that can achieve the required MIPS for processing at least one stream unit according to the value of data processing requirement of such service. Therefore, the user in that case can either reduce the value of minimum stream unit (leading to reduction in the value of required MIPS for processing one stream unit) or add VM offer that satisfies processing at least one stream unit for this service.

Fig. 4 presents the flow of communication for initializing IoTSim-Stream, provisioning SVMs and scheduling ServiceCloudlet on the provisioned SVMs. Once a stream graph application is submitted, GraphAppEngine handles this submission and sends to itself START_DELAY event to allow enough time for BigDatacenters to initialize. During processing this event, GraphAppEngine sends RESOURCE_CHARACTERISTICS event to each BigDatacenter and waiting for their replies. When all BigDatacenters send their replies as RESOURCE_CHARACTERISTICS events, GraphAppEngine processes them and then triggers the process of provisioning and scheduling such application by sending to itself DO_PROVISIONING_AND_SCHEDULING event. In doProvisioningAndScheduling() procedure, the following functions are performed:

1. call collectVMOffers() procedure to collect all VM offers provided by BigDatacenters by querying them.
2. send XML file of submitted application along with the list of VM offers to scheduling policy. This policy then executes processDagFileAndScheduling() procedure to parse this file, extracts the structure of application, selects the best suitable SVMs and prepares the scheduling plan. After the selection of suitable VMs, the objects for SVM and ServiceCloudlet are created.
3. retrieve the generated scheduling plan or table.
4. use this scheduling plan to provision and create SVMs by sending messages (VM_CREATE_ACK) to corresponding BigDatacenters via event mechanism.

While receiving acknowledgements (i.e. VM_CREATE_ACK events) for SVM creations from BigDatacenttres, each acknowledgement for one SVM is processed as it arrives and the corresponding ServiceCloudlet is dispatched to this SVM by calling dispatchServiceCloudlets() procedure; this procedure sends CLOUDLET_SUBMIT event to corresponding BigDatacenter, which processes the received event (CLOUDLET_SUBMIT) and schedules this ServiceCloudlet on a SVM.

Fig. 5 shows the process of sending data streams from external sources and transferring input and output data streams to and from SVMs. Once a stream graph application is being scheduled on SVMs (i.e. ServiceCloudlets of application services have been scheduled on SVMs and ready for execution), the GraphAppEngine sends to itself END_OF_SIMULATION event with the delay specified by user-defined requested simulation time; this event will being sent after this delay, which triggers the end of simulation process. Then, it sends SEND_STREAM events to all external sources requesting them to start sending their data streams to corresponding BigDatacenters, where these datacentres will forward those streams to respective SVMs. At that time, the simulation begins.

Algorithm 1: Simple Provisioning and Scheduling Policy.

Require: minDPUnit a data processing rate for minimum stream unit in an application

for each service in Services **do**
  $selectedVMs \leftarrow \phi$
  $requiredMIPS \leftarrow service.userreq * service.dataprocessingreq$
  $placementCloud \leftarrow$ pick random Cloud from avaliable Clouds
  $VMOffers \leftarrow$ get VM flavours in accending order of power
  for each vm_i in $VMOffers$ **do**
    $vmMIPS \leftarrow$ get vm_i power
    if $vmMIPS/service.dataprocessingreq < minDPUnit$ **then**
      continue
    end if
    if $vmMIPS \leq requiredMIPS$ **then**
      if $i + 1 < n$ **then**
        $nextvmMIPS \leftarrow$ get vm_{i+1} power
        if $nextvmMIPS > requiredMIPS$ **then**
          $toProvisionVM \leftarrow true$
        end if
      else
        $selectedVMs \leftarrow selectedVMs \cup vm_i$
        $requiredMIPS \leftarrow requiredMIPS - vm_i$ power
        $i \leftarrow i - 1$
      end if
    else
      if $i - 1 \geq 0$ **then**
        $previousVmMIPS \leftarrow$ get vm_{i-1} power
        if $previousVmMIPS \geq requiredMIPS$ &&
          $previousVmMIPS < vmMIPS$ &&
        $previousVmMIPS/service.dataprocessingreq \geq minDPUnit$ **then**
          $i \leftarrow i - 2$
        else
          $toProvisionVM \leftarrow true$
        end if
      else
        $toProvisionVM \leftarrow true$
      end if
    end if
    if $toProvisionVM == true$ **then**
      $selectedVMs \leftarrow selectedVMs \cup vm_i$
      $requiredMIPS \leftarrow requiredMIPS - vm_i$ power
      $toProvisionVM \leftarrow false$
    end if
    if $requiredMIPS \leq 0$ **then**
      break
    end if
  end for
  if $selectedVMs$ is empty **then**
    show message 'provisioning failed' to the user
    terminate the currently running simulator (i.e exit)
  end if
end for

Each external source that receives SEND_STREAM event will process it and queries StreamSchedulingOnSVMs object about the portions of its stream and the information of BigDatacenters and SVMs where these portions should be transferred and available. When these portions are received along with the relevant information (i.e. destination BigDatacenters and SVMs), this external source immediately sends them as EXSOURCE_STREAM events to destination BigDatacenters. Each EXSOURCE_STREAM event will be processed by corresponding BigDatacenter whose will send to itself STREAM_AVAILABLE event. It then processes this event to make stream portion available in the corresponding SVM by adding such portion to the input queue of corresponding SVM and sends to itself VM_DATACENTER_EVENT.

When VM_DATACENTER_EVENT being received by BigDatacenter, it processes this event and then updates the state of all simulated entities in a BigDatacenter. At this point, all stream portions available in input queues of all SVMs in all hosts will be moved to the input queues of corresponding ServiceCloudlets via their schedulers, making them available for processing. As well, all output streams available in output queues of ServiceCloudlets as results of computations will be moved to output queues of corresponding SVMs in order to be transferred later. Next, this BigDatacenter sends another VM_DATACENTER_EVENT

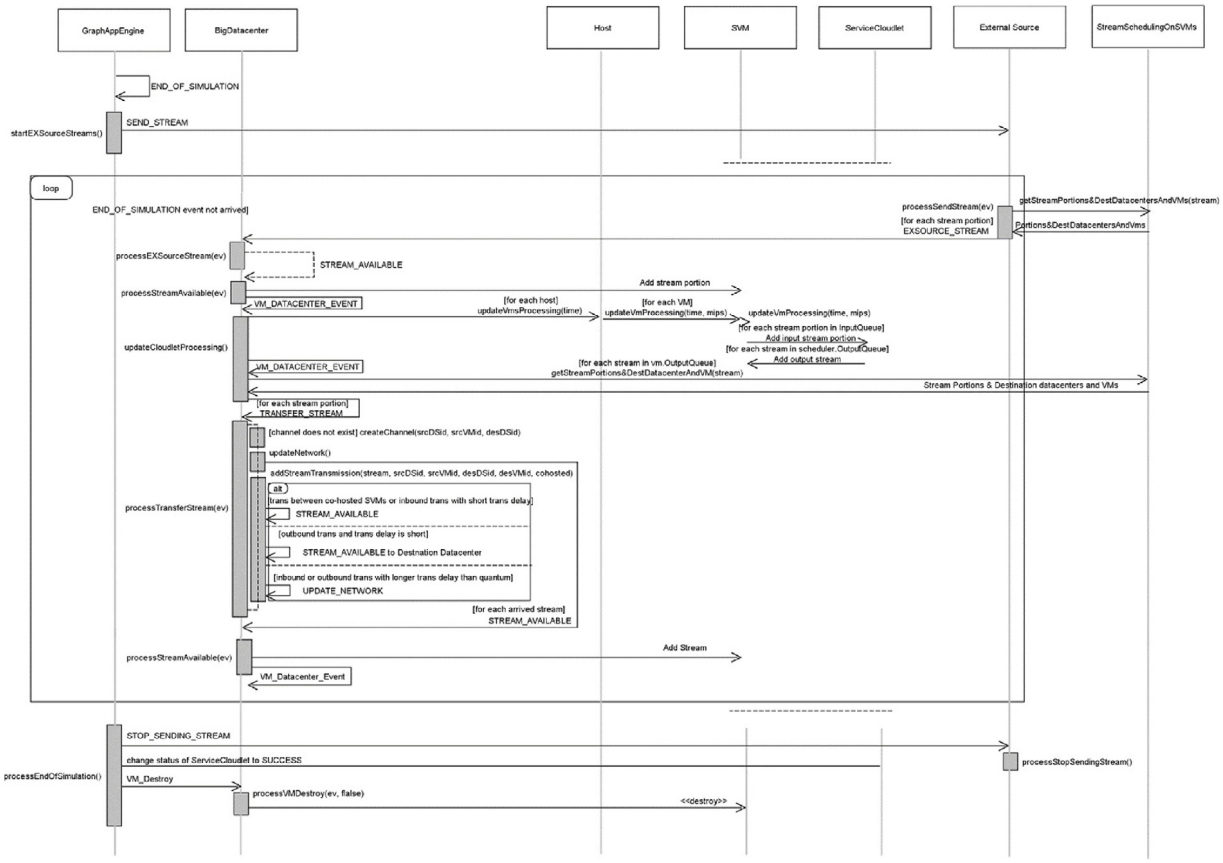


Fig. 5. Sequence Diagram: Transferring and exchanging data streams among SVMs.

to itself for future updating. After that BigDatacenter starts the next communication flow to transfer output streams of ServiceCloudlets available at their SVMs to destination SVMs in order to be input streams for others ServiceCloudlets. Thus, BigDatacenter checks output queues of all hosted SVMs looking for any output stream available as a result of completed computation. For each output stream available at a SVM, it queries StreamSchedulingOnSVMs object about the portions of such stream and the information of destination BigDatacenters and SVMs where these portions should be available. It then use this information to send each stream portion to destination BigDatacenter as a message (i.e. TRANSFER_STREAM event) via event mechanism.

Each TRANSFER_STREAM event is processed by corresponding BigDatacenter whose creates an ingress or egress channel based on whether the transmission of included stream portion is inbound or outbound if such channel does not exist. It then updates its network and adds a new stream transmission to such channel for transferring stream portion. During the addition, if such transmission is between co-hosted SVMs or it is inbound transmission with short transmission delay, this BigDatacenter sends to itself STREAM_AVAILABLE event with cohosted delay for cohosted transmission or with ingress latency for inbound transmission. While if the transmission is outbound transmission and transmission delay is short, this BigDatacenter sends STREAM_AVAILABLE event to the destination BigDatacenter with egress latency to such BigDatacenter. Whereas in case of transmission delay for either inbound or outbound transmission is longer than the pre-defined minimum quantum of time between events (i.e. 0.01 s - 10 ms), this BigDatacenter sends to itself UPDATE_NETWORK event with this delay. Furthermore in network update, this BigDatacenter updates the processing of stream transmissions in all ingress and egress channels, where for each arrived stream, it sends STREAM_AVAILABLE event with ingress or egress latency based on transmission type to corresponding destination BigDatacenter (i.e. itself or other BigDatacenter). Such Bigdatacenter will process this event to make the transferred stream portion available into the corresponding SVM.

Nevertheless, the whole process of transferring and exchanging streams among different SVMs hosted in different BigDatacenters continues until END_OF_SIMULATION event being received (i.e. thereafter the pre-defined delay at the begin of simulation). At that time, GraphAppEngine receives this event and processes it, and then starts the end of simulation process, which includes the followings:

1. stop external sources from sending their streams to corresponding BigDatacenters by sending STOP_SENDING_STREAM events to them.
2. change the status of all ServiceCloudlets to ‘Success’, indicating the end of their executions.
3. destroy all provisioned SVMs by sending VM_Destroy events to their BigDatacenters, which process these events and destroy the hosted SVMs.

When dealing with scheduling, CloudSim has two schedulers, which are VmScheduler and CloudletScheduler. The VmScheduler is host-level scheduler that can run either in space-shared or time-shared mode for allocating cores of processor from a host to VMs (i.e. virtual machine monitor allocation policy). While, the CloudletScheduler is VM-level scheduler that can also run in one of the aforementioned modes for determining the computing power share between Cloudlets in a VM [4]. Since each ServiceCloudlet is submitted to one SVM and this SVM needs to handle the continuous execution of this cloudlet to process incoming streams and produce output stream, the new VM-level scheduler

is required. Therefore, we implement VM-level scheduler named ServiceCloudletScheduler for each SVM within IoTSim-Stream. This scheduler runs in space-shared scheduling mode.

As the ServiceCloudletScheduler is running, it continuously checks its input queue (inputQueue) looking for any incoming streams. If inputQueue is not empty and the waitingStreamsForNextPC flag is true (see Line 37), the ServiceCloudletScheduler enters into the while-loop and performs the following steps on each iteration (see Line 39–57):

1. Fetches the head of inputQueue (i.e. input stream portion with least portion id) and dequeues this stream in case of this stream is not existing in working input stream list (workfingInputStream), and then adding it into such list, preparing for next PC (see Line 42–46).
2. Checks if all required stream portions for one PC arrive and they are added to workfingInputStream in order to perform the appropriate action (see Line 47–56):
   - If yes, it then checks if the time required to process streams included in this PC based on the capacity of cloudlet is less than 0.1, therefore it moves those streams to assumeProcessedStreams list and empties workfingInputStream list. Otherwise, it changes the flag of "check" to false. This flag helps to get out of while-loop either in case of stream portions included in workfingInputStream list need processing time at least as long as the minimum time for one PC (i.e. 0.1) or the head of inputQueue is stream portion for next PC based on portion id.
   - If no, it changes the "check" flag to false if the value of continueCheck flag is false. The continueCheck flag is used to continue in while-loop as the previous head of inputQueue has been dequeued from this queue (see Line 42–46), so that in next iteration, the next head can be fetched and checked to be either dequeued or not. That is very important to fetch and dequeue all of those streams required for one PC as they arrive and before the next update of the scheduler if possible, ensuring low-latency data processing.

When all required stream portions for one PC arrive and they are added to workfingInputStream, and waitingStreamsForNextPC flag is true, the scheduler calculates the total size of input stream portions and using it with the value of data processing requirement for a service to update the length of cloudlet. Then, it changes the startPC flag to true that indicates the start of one PC and waitingStreamsForNextPC flag to false as we are in the phase of starting the execution of one PC (see Line 60–70). After that, the scheduler starts the execution of this PC to process the included stream portions in such PC and updates completion/progress accordingly (see Line 10–13). While the execution of one PC and updating its progress, the scheduler also checks the completion of this PC, so that when the execution finishes (i.e. renaming cloudlet length equals zero), it performs the following steps (see Line 15–35):

1. changes the startPC flag to false that indicates the end of execution of one PC (this PC).
2. produces the output stream and add this stream into outputQueue.
3. empties the working stream list (workfingInputStream)
4. changes the waitingStreamsForNextPC flag to true that indicates the current status backs to wait for stream portions to be arrived if they are not arrived yet and to fetch those portions from input queue required to start new PC.

Algorithm 2: ServiceCloudletScheduler for scheduling and executing ServiceCloudlet on a VM.

```
1: outputQueue ← ϕ
2: inputQueue ← ϕ        ▷ PriorityQueue sorting stream portions by ids - ascending order
3: workingInputStreams ← ϕ        ▷ list of input streams for a Processing Cycle (PC)
4: assumeProcessedStreams ← ϕ        ▷ list of input streams that is assumed to be processed
5: startPC ← false        ▷ flag for starting one PC
6: waitingStreamsForNextPC ← true
7: totalOutputSize ← 0
8: totalInputSize ← 0
9: for each ServiceCloudlet cl in CloudletExecList do        ▷ One ServiceCloudelt exists
10:    if startPC == true then        ▷ when all required input stream portions are available for one PC
11:        start processing stream portions in this cycle
12:        update the completion of this cycle
13:    end if
14:
15:    if waitingStreamsForNextPC == false then        ▷ Execution of one PC for ServiceCloudlet is in progress
16:        if rcl.getRemainingCloudletLength() == 0 then        ▷ Completion of one PC
17:            startPC ← false
18:            produce output stream
19:            if totaOutputSize == 0 then
20:                totaOutputSize ← size of output stream
21:            end if
22:            if totalInputSize == 0 then
23:                totalInputSize ← sum sizes of required input streams
24:            end if
25:            numOfProcessedStreams ← size of workingInputStreams + size of sassumeProcessedStreams size
26:            processedPortionsSize ← max portion size * numOfProcessedStreams
27:            proportionInToOut ← totalOutputSize/totalInputSize
28:            outputStreamSize ← processedPortionsSize ∗ proportionInToOut
29:            create output stream with outputStreamSize
30:            enqueue created output stream in outputQueue
31:            workingInputStreams ← ϕ
32:            assumeProcessedStreams ← ϕ
33:            waitingStreamsForNextPC ← true
34:        end if
35:    end if
36:
37:    if inputQueue is not empty && waitingStreamsForNextPC == true then
38:        check ← true
39:        while check && inputQueue is not empty do
40:            continueCheck ← false
41:            stream_portion ← retrieve the head stream portion of this queue        ▷ not dequeue from priority queue
42:            if stream_portion is not in workingInputStreams then
43:                stream_portion ← perofrm dequeue operation from inputQueue
44:                add stream_portion in workingInputStreams
45:                continueCheck ← true
46:            end if
47:            if stream portions required for one PC being arrived then
48:                if required MIPS for processing these streams in this PC / cloudlet capacity < 0.1 then        ▷ 0.1 is min. time for one PC
49:                    move stream portions to assumeProcessedStreams list
50:                    workingInputStreams ← ϕ
51:                else
52:                    check ← false
53:                end if
54:            else if continueCheck == false then
55:                check ← false
56:            end if
57:        end while
58:    end if
59:
60:    if stream portions required for one PC being arrived && waitingStreamsForNextPC == true then
61:        InPortionsSize ← sum sizes of input stream portions
62:        clLength ← service_processing_req ∗ InPortionsSize        ▷ length of ServiceCloudlet
63:        if cloudlet length == 1 then        ▷ value assigned when cloudlet initialized
64:            cl.length = ((current total length of ServiceCloudlet + clLength) / cpus) -1        ▷ length in MIPS
65:        else
66:            cl.length = (current total length of ServiceCloudlet + clLength) / cpus        ▷ length in MIPS
67:        end if
68:        startPC ← true
69:        waitingStreamsForNextPC ← false
70:    end if
71: end for
```

Table 3
Configuration of datacenters.

Parameter configuration	Datacenter 0	Datacenter 1
Hosts	1000	1000
PEs	64	64
MIPS per PE	1000	2000
RAM per Host (MB)	144000	176000
Storage per Host (MB)	1400000	1500000
VM Boot Delay Time (sec)	20	20

Table 4
Types and configuration of VMs in modelled datacenters.

VM type	Datacenter 0	Datacenter 1
Small	PEs: 2 MIPS: 1000 RAM (MB): 4096 Storage (MB): 8192 Bandwidth (MB/s): 1000	PEs: 2 MIPS: 2000 RAM (MB): 4096 Storage (MB): 8192 Bandwidth (MB/s): 1000
Medium	PEs: 4 MIPS: 1000 RAM (MB): 7168 Storage (MB): 16384 Bandwidth (MB/s): 1000	PEs: 4 MIPS: 2000 RAM (MB): 8192 Storage (MB): 18432 Bandwidth (MB/s): 1000
Large	PEs: 8 MIPS: 1000 RAM (MB): 14336 Storage (MB): 32768 Bandwidth (MB/s): 1000	PEs: 8 MIPS: 2000 RAM (MB): 16384 Storage (MB): 34816 Bandwidth (MB/s): 1000
Extra Large	PEs: 16 MIPS: 1000 RAM (MB): 30720 Storage (MB): 65536 Bandwidth (MB/s): 1000	PEs: 16 MIPS: 2000 RAM (MB): 32768 Storage (MB): 69632 Bandwidth (MB/s): 1000

7. Validation and evaluation

To validate and quantify the efficiency of IoTSim-Stream in simulating stream graph applications in Multicloud environment, we design two experiments, which are simulator validation, and performance and scalability evaluation. We conduct these experiments on a machine that had Intel Core i7-6600U 2.60 GHz (with 2 cores and 4 logical processors), 16GB of RAM memory and running Windows 10 Enterprise, and then collecting the experimental results. In this section, we present our experimental methodology (including Multicloud environment configuration, network configuration, simulation configuration parameters and evaluation experiments) and discusses the experimental results.

7.1. Multicloud environment

Multicloud environment consolidates multiple Clouds in order to maximize the benefits from Cloud services, which opens the door towards orchestrating the execution of multiple applications over various Clouds. To model this environment for our experiments, we define two Clouds (i.e. two Cloud-based datacentres) and configure them as listed in Table 3. For each datacentre, we define four different flavours of VMs, which are Small, Medium, Large and Extra Large, where the configurations of VM vary from one datacentre to another, matching what the Cloud datacentre is in real. Table 4 shows the configurations of VM for the both defined datacentres. This Multicloud environment configuration is consistent throughout the entire evaluation.

7.2. Network configuration

Network performance of Multicloud environment determines the amount of data being transferred within the Cloud-based datacentre (ingress traffic) and between different Cloud-based datacentres (egress traffic). For our experiments, we have conducted TCP bandwidth and latency tests between different zones of Nectar Cloud[1] [18] using IPerf[2] and Ping utility, and then collected the results for both bandwidth (in MB/s) and latency (in second). We chosen average values to model network performance for both ingress and egress traffic for Cloud-based datacentres in the modelled Multicloud environment as listed in Table 5. Since studying the network performance is out of scope of this paper and for simplicity purpose, we made the configuration of network performance for both Cloud-based datacentres in the modelled environment is identical with slight difference. This configuration of network performance for those datacentres is consistent throughout the entire evaluation.

Table 5
Configuration of network performance of modelled multicloud environment.

Network parameter	Datacenter 0	Datacenter 1
Ingress Bandwidth	770 MB/s	780 MB/s
Ingress Latency	0.00077 s	0.00075 s
Egress Bandwidth	170 MB/s	180 MB/s
Egress Latency	0.028 s	0.026 s

7.3. Simulation configuration properties

Prior to run the simulator, we need to configuration its parameters that are defined in simulation properties file (simulation.properties). These parameters will be read by IoTSim-Stream during initialization for preparing to simulate given stream graph application according to specified configurations. Table 6 shows the simulation parameters that included in this file with their description and values used in our experiments.

The parameters from “cloud.provider” to “external.latency” shown in the above table need to be repeated for each Cloud provider (i.e Cloud-based datacentre) defined in Multicloud environment. As we mentioned earlier for our experiments, we define and configure two datacentres as listed in Table 3. Thus, two sets of these parameters are defined in simulation.properties file, where the first set is for the first datacentre and the second set is for the second datacentre.

7.4. Evaluation experiments

As we mentioned earlier, two experiments are considered for our evaluation of IoTSim-Stream, which are as follows:

- Experiment 1 (Simulator Validation): Validate the correctness of IoTSim-Stream in modelling, scheduling and executing stream graph applications in Multicloud environment. This experiment presents a comparison between the amount of data streams being processed in simulated and real time (theoretical) executions for different structures of stream graph applications. Theoretical execution is a manual (hand-held) process to execute stream graph application service-by-service and collect the results for total amount of data streams being processed by this application, providing a rigorous results to compare with simulated results. Using these applications in their simple form (called simple stream graph applications) rather than their complex form (called complex stream graph applications) in this comparison is sufficient. That is because conducting this comparison for complex stream graph applications is not only so complicated in real time using theoretical execution, but it adds

[1] The National eResearch Collaboration Tools and Resources project (Nectar) provides Cloud computing infrastructure for Australia's research community.

[2] IPerf is a cross-platform network performance measurement tool for both Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

Table 6
User-defined simulation parameters configuration.

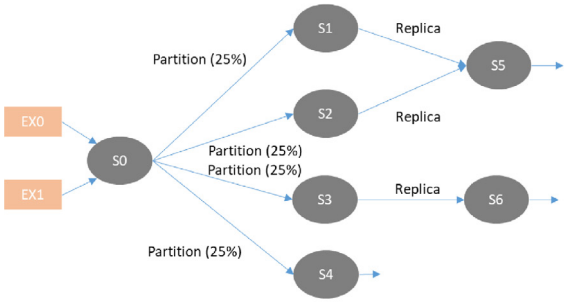
Parameter	Description	Value
simulation time	The requested simulation time in seconds	Refer to experiments
scheduling.policy	Provisioning and scheduling policy	*SimpleSchedulingPolicy*
dag.file	Path of XML file of stream graph application	*path_value*
cloud.datacenter	Number of Clouds in Multicloud environment, where each Cloud is represented by a datacenter	2
engine.network.bandwidth	Network bandwidth of GraphAppEngine	300
engine.network.latency	Network latency of GraphAppEngine	0.05
cloud.provider	Index of Cloud provider in Multicloud environment (index starting from 0)	*value*
datacenter.hosts#*index* (ex. datacenter.hosts#0)	number of hosts in datacenter	*value*
vm.delay#index	Average delay of VM boot time	*value*
vm.offers#index	Path of Java class for offerings of Cloud-based datacentre	*packagename.classname*
host.cores#index	Number of cores (PEs) available for each host	*value*
host.memory#index	Amount of memory available for each host	*value (unit: MB)*
host.storage#index	Amount of storage available for each host	*value (unit: MB)*
core.mips#index	MIPS for each core or PE	*value*
internal.bandwidth#index	Internal network bandwidth available for each VM within Cloud-based datacentre	*value (unit: MB/s)*
internal.latency#index	Network delay between VMs within Cloud-based datacentre	*value (unit: MB/s)*
external.bandwidth#index	External network bandwidth available by Cloud-based datacentre for transferring data streams to other datacentres	*value (unit: MB/s)*
external.latency#index	Network delay from Cloud-based datacentre to other dataentres	*value (unit: MB/s)*

SERVICE / EXTERNAL SOURCE	DATA PROCESSING REQUIREMENT	USER PERFORMANCE REQUIREMENT	INPUT	OUTPUT	OUTPUT STREAM TO SERVICE(S)
EX0	N/A	N/A	N/A	4 MB/s	4 MB/s to S0
S0	500 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S1
S1	500 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S2
S2	1000 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S3
S3	1000 MI/MB	4 MB/s	4 MB/s	1 MB/s	None

(a) Structure and configuration of stream graph application 1 (App1)

SERVICE / EXTERNAL SOURCE	DATA PROCESSING REQUIREMENT	USER PERFORMANCE REQUIREMENT	INPUT	OUTPUT	OUTPUT STREAM TO SERVICE(S)
EX0	N/A	N/A	N/A	4 MB/s	4 MB/s to S0
S0	500 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S1 4 MB/s to S2
S1	500 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S3
S2	1000 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S3
S3	1000 MI/MB	8 MB/s	8 MB/s	1 MB/s	None

(b) Structure and configuration of stream graph application 2 (App2)



SERVICE / EXTERNAL SOURCE	DATA PROCESSING REQUIREMENT	USER PERFORMANCE REQUIREMENT	INPUT	OUTPUT	OUTPUT STREAM TO SERVICE(S)
EX0	N/A	N/A	N/A	2 MB/s	2 MB/s to S0
EX1	N/A	N/A	N/A	2 MB/s	2 MB/s to S0
S0	2000 MI/MB	4 MB/s	4 MB/s	16 MB/s	4 MB/s to S1 4 MB/s to S2 4 MB/s to S3 4 MB/s to S4
S1	1000 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S5
S2	2000 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S5
S3	1000 MI/MB	4 MB/s	4 MB/s	4 MB/s	4 MB/s to S6
S4	4000 MI/MB	4 MB/s	4 MB/s	1 MB/s	None
S5	4000 MI/MB	8 MB/s	8 MB/s	2 MB/s	None
S6	4000 MI/MB	4 MB/s	4 MB/s	1 MB/s	None

(c) Structure and configuration of stream graph application 3 (App3)

Fig. 6. Stream graph applications with their parameter configurations for our experiments.

more complication without any need or benefit in such validation. Thus from this comparison, we can ensure the correctness of modelling simple stream graph applications in Cloud infrastructures, inferring to the correctness of modelling even more complex stream graph applications as well, since only the complexity of application structure and its performance requirements are what varies.

- Experiment 2 (Performance and Scalability Evaluation): Study of the performance of IoTSim-Stream in term of execution time, CPU and memory usage along with the total amount of processed data streams with small to medium to extremely large stream graph applications. This experiment shows the ability of proposed simulator to model, simulate and schedule not only simple stream graph applications, but

Table 7
Mapping services of stream graph applications on VMs.

	APP1	APP2	APP3
Required MIPS per Service	S0: 2000 S1: 2000 S2: 4000 S3: 4000	S0: 2000 S1: 2000 S2: 4000 S3: 8000	S0: 8000 S1: 4000 S2: 8000 S3: 4000 S4: 16000 S5: 32000 S6: 16000
VM Type per Service	S0: Small - Datacenter 0 S1: Small - Datacenter 0 S2: Small - Datacenter 1 S3: Small - Datacenter 1	S0: Small - Datacenter 0 S1: Small - Datacenter 0 S2: Small - Datacenter 1 S3: Medium - Datacenter 1	S0: Medium - Datacenter 1 S1: Medium - Datacenter 0 S2: Large - Datacenter 0 S3: Medium - Datacenter 0 S4: Large - Datacenter 1 S5: Extra Large - Datacenter 1 S6: Extra Large - Datacenter 0

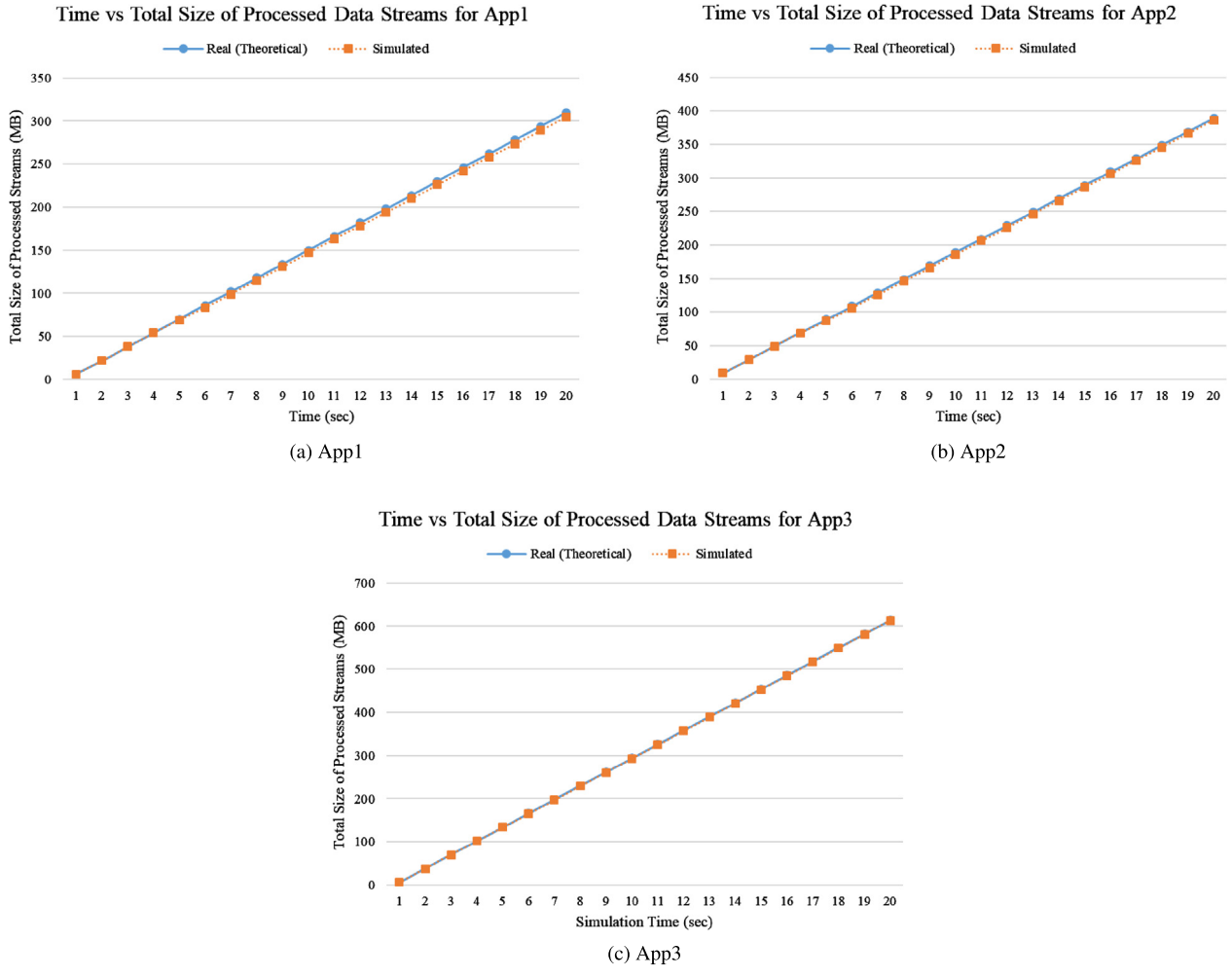


Fig. 7. Real and simulated total size of processed data streams of three graph applications.

even more complex stream graph applications in Multicloud environment. That makes researchers confidently study the behaviours of different structures and configuration sizes of stream graph applications for further evaluations and improvements. For example, developing new provisioning and scheduling policies, improving execution performance, and studying QoS and SLA requirements for this type of applications.

Fig. 6 shows the structures and parameter configurations of three stream graph applications (named App1, App2 and App3) that will be used in our experiments. For Experiment 1, we use those modelled applications in their simple form as shown in this figure. While for Experiment 2, we use them in their complex form (i.e. each one of them is replicated several times to generate the complex graph structure with hundreds and thousands of nodes (services)) to assess the performance and scalability of IoTSim-Stream. As seen from this figure, each stream graph application is composed of multiple services with one or more external sources. Each external source produces output data stream per second according to its data rate (in MB/s) that will be feed into corresponding service(s). And each service in this application

needs the following configurations: data processing requirement, user performance requirement, input streams and output stream.

7.5. Experiment 1: validation

To validate the behaviour of IoTSim-Stream, two tests are conducted. In the first test, we undertook the theoretical execution of the three modelled stream graph applications for 20 s and collect the total size pf processed data streams as experimental results for this real execution. While in the second test, we undertook the simulated execution of these applications on real Cloud infrastructure using IoTSim-Stream for also 20 s and collect the total size of processed data streams as experimental results. In these experiment tests, we use the default value of data processing rate for minimum stream unit (i.e 1MB/s) defined in IoTSim-Stream for both real and simulated executions. Thus, any stream that is larger than minimum stream unit will be divided into portions and each portion is 1MB in size. For example, if the input rate of service is 4MB/s, the stream will be divided into four portions. As well for this experiment, we pre-defined the mapping of services of modelled applications (App1, App2 and App3) on VMs, where each service has one ServiceCloudlet mapped on one VM as listed in Table 7. Of course, there are many possible VM mappings of services of these applications, but we only present one VM mapping and use it in these experiment tests.

As comparing the total amount of data streams being processed by each modelled application in given time is a real indication for measuring the accuracy, we then compare the collected experimental results of both real and simulated executions in order to quantify the accuracy and precision of IoTSim-Stream . Fig. 7 shows the real and simulation results for modelled stream graph applications. Certainly, the increase in time leads to an increase in the amount of data streams being processed by a stream graph application. The difference between both results is very slight and the results of IoTSim-Stream simulation match very closely to the real ones. As the time increases, the little difference occurred between both results is being reduced and the simulation results become more closer to match real ones. Consequently, the accuracy of simulation results from IoTSim-Stream in comparison with theoretical results is indicated that IoTSim-Stream is efficient in modelling and simulating the execution of different structures of stream graph applications on real Multicloud environment.

7.6. Experiment 2: performance and scalability evaluation

As we mentioned before, the aim of this experiment is to analyse the overhead and scalability of CPU and memory usages as well as measuring the execution time of IoTSim-Stream simulations along with the total amount of data streams being processed during these simulations. Thus in this experiment, we use the modelled stream graph applications (App1, App2 and App3) with varying configuration sizes (ranging from very small to extremely large) as listed in Table 8. Each configuration size has different number of services and DPs.

The CPU usage information is collected using built-in Java management interface for the operating system (called "OperatingSystemMXBean") on which the Java Virtual Machine (JVM) is running. This usage is measured every second during simulation time and the average value is taken. While the memory usage information is collected using Java Runtime. The execution time is the time required to simulate given application at a given simulation time. Each test was repeated 10 times and average results are obtained and used in representation of experimental results. The provisioning and scheduling policy presented in Algorithm 1 is used to schedule each configuration size of each application on SVMs, where the scheduling plan for each one is the same across all ten repeated simulations. The default value of data processing rate for minimum stream unit (i.e 1MB/s) defined in IoTSim-Stream is also used in this experiment.

7.6.1. Experimental tests under fixed simulation time

The first set of tests are aimed at evaluating performance and scalability of IoTSim-Stream with different configuration sizes of the modelled applications when the simulation time is set to 5 min. Prior to analysis the obtained performance and scalability results, it is worth discussing the experimental results for the total amount of data streams being processed by modelled applications with their different configuration sizes. This discussion gives an indication about the amount of computations that carried-out and helps to quantify the performance of IoTSim-Stream by magnitude of processed data streams. Fig. 8 shows the experimental results for total size of data streams being processed by each configuration size of each modelled application. From this figure, it is clear that as the configuration size of application increases the amount of processed streams is increasing, where the total size of processed streams reaches about 3TB with App2_doublelarge and App3_doublelarge for 5 min simulation. The exception from this increasing is App1_doublelarge since this application is linear and replicating it is also in linear way, and as simulation time is set to 5 min, the additional 1000 services from the prior configuration size did not process any streams (i.e. they are waiting for them). Therefore, the total amount of processed streams for this configuration size is the same as App1_verylarge.

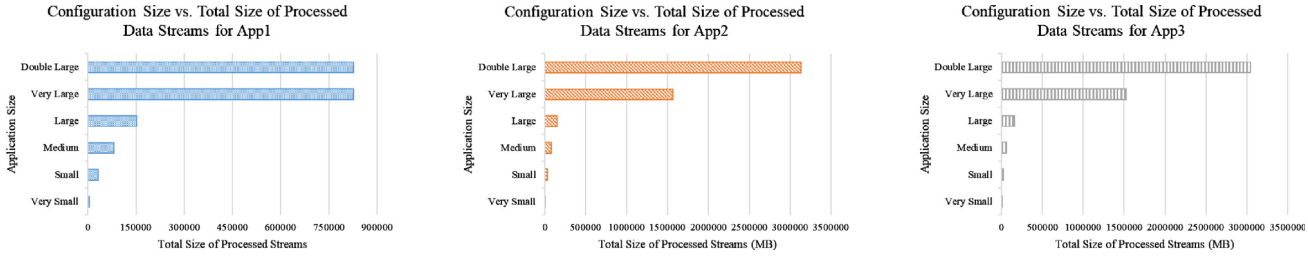
Another point from Fig. 8 is that the total amount of streams processed by App3 in comparison with App2 is approximately the same in some cases and less in other cases particularly from small configuration size, even though App3 has a close or even more number of services and its parameter configurations shown in Fig. 6 indicated that the total amount of streams being processed per second by its services according to user performance requirements is also greater. The reason behind it is that by considering the number of services of this application (i.e. 7 services), the replication of App3 several times to reach the number of services required at each configuration size makes the total number of services being replicated is less than those services being replicated in App2, where some more intermediate and final merging services are needed to merge outputs of replicated services and produce outputs as original application. For example, to generate App3_small and App3_verylarge, App2 services are replicated 2 times and 141 times (i.e. # of services be 14 and 987) respectively, while to generate App2_small and App2_verylarge, App2 services are replicated 5 times and 248 times (i.e. # of services be 20 and 992) respectively, and the rest service(s) is/are added as intermediate and final merging services (i.e. 3 for App3_small, 15 for App3_verylarge, 1 for App2_small, and 9 for App2_Large). Overall, the amount of data being processed by those applications is huge and IoTSim-Stream is simulating them effectively.

The performance and scalability results for modelled stream graph applications with their different configuration sizes are depicted in Fig. 9. From the experimental results shown in this figure, our analysis and findings are summarized as follows:

- The results of execution time showed that the execution time is slightly increasing from very small to large configuration sizes with all modelled applications, where IoTSim-Stream is able to simulate large configuration size of App1, App2 and App3 for 5 min in approximately 6 s, 5 s and 5 s respectively and using less than 560MB of memory, where the total size of processed streams is approximately 149GB by App1, 148GB by App2 and 157GB by App3. While for very large and double large configuration sizes of the modelled applications, the execution time is significantly increased.

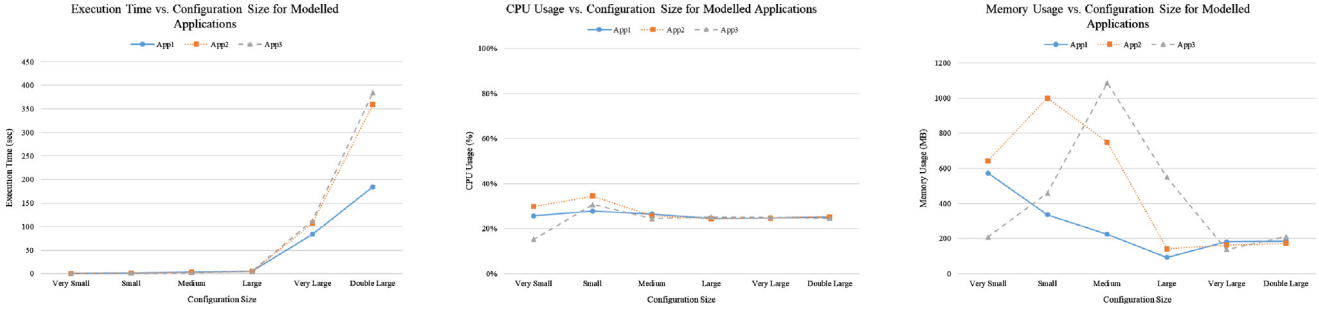
Table 8
Number of services and DPs in each configuration size for each modelled stream graph application.

SIZE	APP1	APP2	APP3
Very Small	4 Services - 1 DP (called App1_verysmall)	4 services - 1 DP (called App2_verysmall)	7 services - 2 DPs (called App3_verysmall)
Small	20 services - 5 DPs (called App1_small)	21 services - 5 DPs (called App2_small)	17 services - 4 DPs (called App3_small)
Medium	52 services - 13 DPs (called App1_medium)	53 services - 13 DPs (called App2_medium)	45 services - 12 DPs (called App3_medium)
Large	100 services - 25 DPs (called App1_large)	100 services - 24 DPs (called App2_large)	108 services - 30 DPs (called App3_large)
Very Large	1000 services - 250 DPs (called App1_verylarge)	1001 services - 248 DPs (called App2_verylarge)	1002 services - 282 DPs (called App3_verylarge)
Double Large	2000 services - 500 DPs (called App1_doublelarge)	2001 services - 496 DPs (called App2_doublelarge)	2001 services - 564 DPs (called App3_doublelarge)



(a) App1 (b) App2 (c) App3

Fig. 8. Total size of processed data streams with different configuration sizes of the modelled applications.



(a) Execution Time (b) CPU Usage (c) Memory Usage

Fig. 9. Performance of the modelled applications with different configuration sizes.

This behaviour is expected as the number of services is 10 times and 20 times more than the number of services in large configuration size respectively as well as the total amount of streams being processed by those applications is also sharply increased. As an instance, IoTSim-Stream simulates App2_verylarge and App2_doublelarge for 5 min in approximately 1.8 min and 6 min respectively and using less than 200MB of memory, with total amount of processed streams is approximately 1.5TB by App2_verylarge and 3TB by App2_doublelarge. Thus, IoTSim-Stream is able to simulate a complex stream graph application with thousands of services that process huge amount of data streams (big data) with excellent performance and scalability.

- The results of CPU usage for all modelled applications with all configuration sizes except very small and small configuration sizes is not exceed 27%. This usage is an excellent CPU performance in the machine that has 4 logical processors where this experiment is conducted, which translates to roughly usage of one logical processor. Certainly, more computing power allocated to VM leads to further decline in CPU usage. For CPU usage with very small and small configuration sizes of modelled applications, the percentage is little higher as the simulation of these applications is completed in less than 2.2 s, so that some of measured usages are CPU bursts.
- The results of memory usage showed that memory fluctuates with different configuration sizes of different modelled application. These results also showed that IoTSim-Stream is able to simulate double large configuration size of modelled applications used less than 220MB of memory. These observations proved that IoTSim-Stream is capable of simulating complex stream graph applications with little memory overhead.
- IoTSim-Stream not only provides the ability to simulate different stream graph applications, it also offers significant gains in regards to easily measure and evaluate the execution performance. These gains are very important as it is almost unattainable to calculate and collect the execution time and performance (in term of CPU and memory usage) in a large-scale test environment on Multicloud environment.

7.6.2. Experimental tests under varying of simulation times

The second set of tests are aimed at evaluating performance and scalability of IoTSim-Stream with chosen configuration sizes for the modelled applications when simulation time is varying. For these tests, we chose two configuration sizes to study non-complex and complex structure of the modelled stream graph applications with the following simulation times: 300 (5min), 600 (10min), 1200 (20min), 1800 (30min), 2400 (40min), 3000 (50min) and 3600 (1 h).

Fig. 10 depicts the total amount of streams processed by chosen configuration sizes of the modelled applications. As expected, the amount of processed data streams is increased as simulation time increases for all modelled applications, where the maximum total size of processed streams with small configuration size is approximately 379.6GB for App2, and with very large configuration size is 18TB for App1 in 1 simulation hour. That is showing how the amount of streams being processed is huge and IoTSim-Stream is effectively simulating those applications on Multicloud environment.

Fig. 11 depicts the performance and scalability results of chosen configuration sizes of the modelled applications. The results showed that the execution time with small configuration size of all modelled application except App3 is scaled sub-linearly as simulation time increases, where IoTSim-Stream is completed 1 h of simulation for small configuration size of App1 in less than 17 s, App2 in less than 15 s and App3 in less than 8 s. These performance observations for execution time with small configuration size proved that IoTSim-Stream is effectively simulating those applications for long simulation times in a very short time, within a matter of seconds. For very large configuration size of all modelled application, the results showed that the execution time is scaled sub-linearly as simulation time increases, where IoTSim-Stream is completed 1 h of simulation for very large configuration size of App1 in less than 29 min, App2 in less than 24 min and App3 in less than 27 min. These performance observations for execution time with very large configuration size of modelled applications proved that IoTSim-Stream is effectively simulating those complex applications for long simulation times in a short and reasonable time, within a matter of minutes.

In regards to CPU usage with small configuration size, we observed that a fluctuation between approximately 10% and 41% for modelled applications when simulation time is 5 min. As simulation time increases, we observed a steady usage and this usage is not exceed 26%. While with very large configuration size, we observed a steady usage as simulation time increases and the this usage is not exceed 28%.

As regards memory usage with small configuration size, the results showed a significant dropping in this usage for App1 at 40 min of simulation and slight dropping in this usage for App2 and App3 at 30 min of simulation due to the behaviour of modelled application, and then it becomes steady as simulation time increases with all modelled applications. The lowest memory usage recorded with small configuration size is 36MB. While with very large configuration size, we observed that memory usage is scaled sub-linear and never grew beyond 660 MB even for 1 h of simulation. Therefore, less than 700MB of memory is sufficient for IoTSim-Stream to simulate very large configuration size of each modelled application for 1 h, where each application processed several terabytes of data streams during this simulation.

8. Significance and practicality of IoTSim-stream

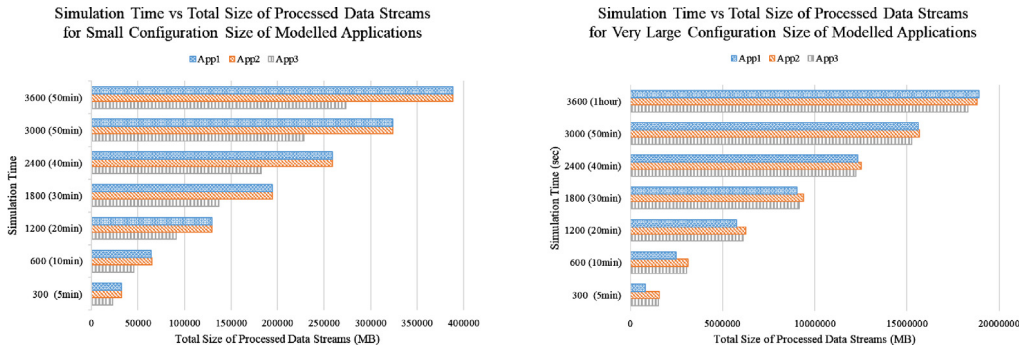
To have a look on the practicality of the proposed simulator, we discuss one of IoT graph applications in smart cities as a real world example. Connected cars application has become largely and widely accepted. By 2020, Gartner foresees more than a quarter billion connected vehicles on the road, where each one of them produces approximately 25GB of data per driving hour [4]. Analysing the flood of data coming from roadside infrastructure (e.g. traffic lights, cameras) and connected cars allow to get real-time analytical insights that help in different services of smart city such as traffic condition and control, and smart parking. Modelling such type of IoT application using IoTSim-Stream is a straightforward task to investigate how this application will behave and evaluate its performance in Cloud infrastructures at no execution cost.

In this IoT graph application, each roadside infrastructure device or connected car can be modelled as an external source, and each analytical component (such as vehicle detection, roadside data analysis, traffic analysis and traffic controlling) can be modelled as an independent service and is executed over any virtual resources. The coordination of application execution (i.e. control flow) and data dependencies (i.e. data flow) among the modelled services are defined in accordance of application logic. Based on that, the flows of data from external sources are continuously injected into the corresponding services and those flows from internal sources as continuous output streams which are results of the continuous computations carried-out by modelled services are routed towards the corresponding services.

The structure of this graph application involves heterogeneous services, multiple data sources, multiple input and output streams, can now be expressed in DAG file by including all modelled services with their data processing requirements and performance constraints that defined by the owner of this application, and data dependencies among them. Moreover, IoTSim-Stream supports the modelling of different patterns/structures of stream workflow applications, which are linear, branching and hybrid. Linear workflow pattern (like App1) is a multi-stage application, where each stage processes input stream generated by the previous stage and produces the output stream to the following stage. Branching workflow pattern (like App2) is an application with limited precedence constraints that splits data stream to perform different parallel processing and then combining the results for further analysing. Hybrid workflow pattern (like App3) is a mix of linear and branching patterns. Thus, whether the pattern/structure of the aforementioned IoT graph application is linear, branching or hybrid with various data processing requirements and configuration complexities, IoTSim-Stream is able to simulate it in Multicloud environment. Furthermore, IoTSim-Stream enables the researcher to define the execution environment with its network performance (i.e. Multicloud environment), providing the full capability to study and investigate the performance of this application in Cloud computing platforms.

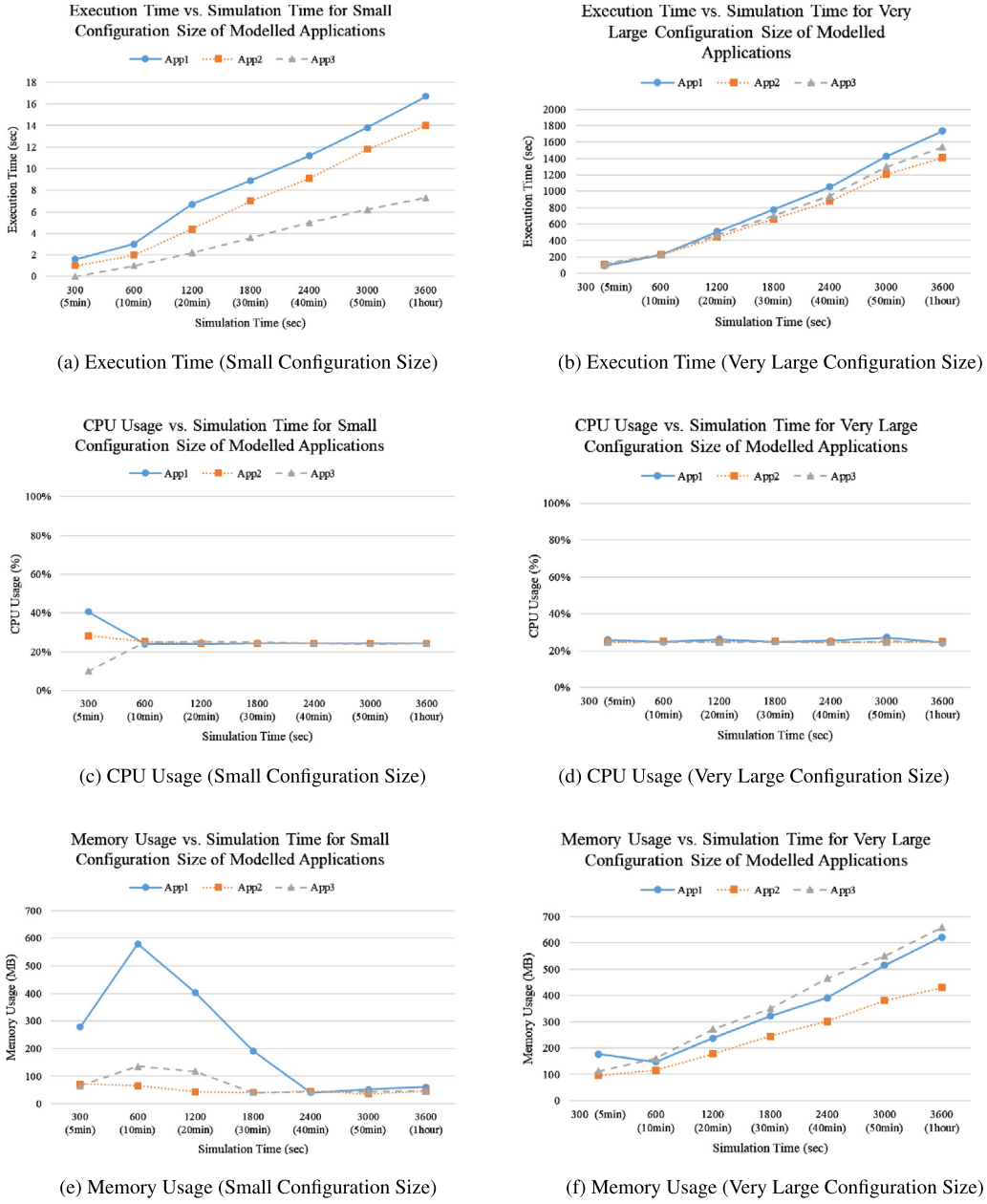
Accordingly, the aforementioned real world example illustrates the need of modelling and orchestrating sIoT graph application in simulation environment to support experiments at planning phase for further enhancing and improving prior to being deployed in real Cloud infrastructures at production phase. By controlling the configurations of graph application, execution environment and simulation environment, the difficulty of hand over the power of real-time data analytics is simplified even with most complicated and distributed data pipelines. Thus, the requirements of achieving real-time data analysis and efficient workflow orchestration can be investigated through controllable and repeatable experiments, leading to further research studies including proposing resource and scheduling policies that adheres to user-defined SLA and QoS requirements, improving performance and minimizing execution cost - that is what the generalized IoTSim-Stream aims to provide.

From the above discussion, our proposed simulator offers significant benefits to researchers, allowing them to (1) study how



(a) Small Configuration Size (b) Very Large Configuration Size

Fig. 10. Performance evaluation with small and very large configuration sizes of the modelled applications.



(a) Execution Time (Small Configuration Size) (b) Execution Time (Very Large Configuration Size)

(c) CPU Usage (Small Configuration Size) (d) CPU Usage (Very Large Configuration Size)

(e) Memory Usage (Small Configuration Size) (f) Memory Usage (Very Large Configuration Size)

Fig. 11. Performance of the modelled applications with small and very large configuration sizes.

stream graph applications will perform in the Cloud and its performance, (2) evaluate the efficiency of new scheduling and resource allocation policies for such applications in a real-world simulation environment, (3) test SLA-oriented management and execution optimization of stream graph applications in Cloud infrastructures free of cost, and (4) tune the performance bottlenecks at planning and testing stage prior to go production by deploying the stream graph application on multiple commercial Cloud platforms. Furthermore, IoTSim-Stream is designed in mind to be extensible and customizable simulation toolkit, so that it provides the ability for researchers to extend and define their policies for adhering user-defined SLA and QoS requirements, and execution optimization as well as extending and defining policies in all components of CloudSim software stack since it was built on top of CloudSim. As a result, IoTSim-Stream is a right research simulation toolkit that deals with both complexities emerging from modelling stream graph application and simulated environments.

9. Conclusion

In this paper, we proposed IoTSim-Stream, a simulation toolkit for modelling stream graph applications in Multicloud environment. We also presented the main components of IoTSim-Stream with their functionalities. IoTSim-Stream provides fully custom simulation parameters, making it a suitable research tool to assist researchers in simulating and studying the behaviour of stream graph application in Cloud computing environments with easy to set-up Multicloud environment and customizable user performance requirements. From the results of real and simulated experiment, IoTSim-Stream has been validated and its correctness is proven in simulating various structures of stream graph applications. Moreover from the results of extensive performance and scalability evaluations, IoTSim-Stream is proved to be effectiveness in modelling and simulating linear, branching and hybrid patterns/structures of stream graph applications with various data processing requirements and configuration complexities ranging from simple to moderate to even more complex configurations.

As IoTSim-Stream a customizable and extensible simulation toolkit, it enables both industry and research communities to conduct further research studies by extending and defining policies for meeting user-defined SLA and QoS requirements, and for execution optimization in addition to define custom ones in all components of CloudSim software stack, to test the performance of stream graph applications with their policies more accurately in a controlled, repeated and easy to set-up simulation environment.

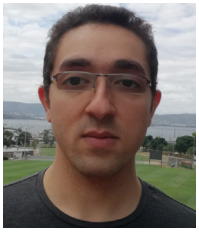
In the future, we will include failure model to simulate the occurrences of failures at service-level or virtual machine-level, and study more service level agreement constraints such as deployment costs, performance. We would also like to support workflow monitoring to monitor the continuous execution of stream graph applications.

Acknowledgement

This research is supported by an Australian Government Research Training Program (RTP) Scholarship.

References

[1] R. Ranjan, S. Garg, A.R. Khoskbar, E. Solaiman, P. James, D. Georgakopoulos, Orchestrating bigdata analysis workflows, IEEE Cloud Comput. 4 (2017) 20–28.
[2] P. Mell, T. Grance, et al., The Nist Definition of Cloud Computing, 2011.
[3] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (2009) 599–616.
[4] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, R. Buyya, Cloudsim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. - Pract. Exp. 41 (2011) 23–50.
[5] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soulé, et al., Ibm streams processing language: Analyzing big data in motion, IBM J. Res. Dev. 57 (2013) 7–1.
[6] H. Hu, Y. Wen, T.S. Chua, X. Li, Toward scalable systems for big data analytics: A technology tutorial, IEEE Access 2 (2014) 652–687.
[7] S.K. Garg, R. Buyya, Networkcloudsim: Modelling parallel applications in cloud simulations, in: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, IEEE, 2011, pp. 105–113.
[8] G. Wang, A.R. Butt, P. Pandey, K. Gupta, Using realistic simulation for performance analysis of mapreduce setups, in: Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance, ACM, 2009, pp. 19–26.
[9] A.C. Murthy, Mumak: Map-Reduce Simulator. MAPREDUCE-728, Apache JIRA, 2009.
[10] H. Tang, Mumak: Map-Reduce Simulator, 2009.
[11] A. Verma, L. Cherkasova, R.H. Campbell, Play it again, simmr!, in: Cluster Computing (CLUSTER), 2011 IEEE International Conference on, IEEE, 2011, pp. 253–261.
[12] S. Hammoud, M. Li, Y. Liu, N.K. Alham, Z. Liu, Mrsim: A discrete event based mapreduce simulator, in: Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on, IEEE, 2010, pp. 2993–2997.
[13] J. Jung, H. Kim, Mr-cloudsim: Designing and implementing mapreduce computing model on cloudsim, in: ICT Convergence (ICTC), 2012 International Conference on, IEEE, 2012, pp. 504–509.
[14] X. Zeng, S.K. Garg, P. Strazdins, P. Jayaraman, D. Georgakopoulos, R. Ranjan, Iotsim: a cloud based simulator for analysing iot applications, 2016, ArXiv preprint arXiv:1602.06488.
[15] W.A. Higashino, M.A. Capretz, L.F. Bittencourt, Cepsim: Modelling and simulation of complex event processing systems in cloud environments, Future Gener. Comput. Syst. 65 (2016) 122–139.
[16] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: E-Science (e-Science), 2012 IEEE 8th International Conference on, IEEE, 2012, pp. 1–8.
[17] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.H. Su, K. Vahi, Characterization of scientific workflows, in: Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on, IEEE, 2008, pp. 1–10.
[18] NectarCloud, Nectar cloud, 2018, URL https://nectar.org.au/.



Mutaz Barika has obtained his BSc. and MSc. in Computer Science from University of Petra and King Saud University respectively. He is currently a Ph.D. Candidate at University of Tasmania, Australia. He has been awarded an Australian Government Research Training Program (RTP) Scholarship for supporting his studies. His current research interests include Big Data, Big Data Workflow, Cloud Computing, IoT and Data Security.

Saurabh Garg is a Lecturer at the University of Tasmania, Hobart, Tasmania. He is one of the few Ph.D. students who completed in less than three years from the University of Melbourne in 2010. He has gained about three years of experience in the Industrial Research while working at IBM Research Australia and India. His area of interests are Distributed Computing, Cloud Computing, HPC, IOT, BigData analytics, and education analytics.

Andrew Chan is the Professor and Head, School of Engineering at the University of Tasmania. He is one of the world leading experts in the use of the finite element method of static and dynamic fully coupled soil and pore-fluid interaction. He also works in Discrete Element Method, Scaled Boundary Finite Element Method and Lattice Boltzmann. He worked in the use of various SIMD and MIMD strategies to speed up analyses with high computational requirement.

Rodrigo N. Calheiros is a Senior Lecturer in the School of Computing, Engineering and Mathematics, Western Sydney University, Australia. He works in the field of Cloud computing and related areas since 2008, and since them he carried out R&D supporting research in the area. His research interests also include Big Data, Internet of Things, Fog Computing, and their application.

Rajiv Ranjan is a Chair and Professor in Computing Science at Newcastle University, United Kingdom, as well as at China University of Geosciences, China. Before moving to Newcastle University, he was Julius Fellow (2013–2015), Senior Research Scientist and Project Leader in the Digital Productivity and Services Flagship of Commonwealth Scientific and Industrial Research Organization (CSIRO — Australian Governments Premier Research Agency). Prior to that he was a Senior Research Associate (Lecturer level B) in the School of Computer Science and Engineering, University of New South Wales (UNSW). He has a Ph.D. (2009) from the department of Computer Science and Software Engineering, the University of Melbourne.